Electronic Colloquium on Computational Complexity, Report No. 45 (2021)

Reconstruction Algorithms for Low-Rank Tensors and Depth-3 Multilinear Circuits.

Vishwas Bhargava* Shubhangi Saraf† Ilya Volkovich ‡

Abstract

We give new and efficient black-box reconstruction algorithms for some classes of depth-3 arithmetic circuits. As a consequence, we obtain the first efficient algorithm for computing the tensor rank and for finding the optimal tensor decomposition as a sum of rank-one tensors when then input is a *constant-rank* tensor. More specifically, we provide efficient learning algorithms that run in randomized polynomial time over general fields and in deterministic polynomial time over $\mathbb{R}$ and $\mathbb{C}$ for the following classes:

1. *Set-multilinear depth-3 circuits of constant top fan-in (*$\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ *circuits).* As a consequence of our algorithm, we obtain the first polynomial time algorithm for tensor rank computation and optimal tensor decomposition of constant-rank tensors. This result holds for d dimensional tensors for any d, but is interesting even for $d = 3$.
2. *Sums of powers of constantly many linear forms (*$\Sigma \wedge \Sigma(k)$ *circuits).* As a consequence we obtain the first polynomial-time algorithm for tensor rank computation and optimal tensor decomposition of constant-rank symmetric tensors.
3. *Multilinear depth-3 circuits of constant top fan-in (multilinear* $\Sigma\Pi\Sigma(k)$ *circuits).* Our algorithm works over all fields of characteristic 0 or large enough characteristic. Prior to our work the only efficient algorithms known were over polynomially-sized finite fields [KS09a].

Prior to our work, the only polynomial-time or even subexponential-time algorithms known (deterministic or randomized) for subclasses of $\Sigma\Pi\Sigma(k)$ circuits that also work over large/infinite fields were for the setting when the top fan-in k is at most 2 [Sin16, Sin20].

*Department of Computer Science, Rutgers University, Piscataway, NJ 08854. Research supported in part by the Simons Collaboration on Algorithms and Geometry and NSF grant CCF-1909683. Email: vishwas1384@gmail.com.

†Department of Mathematics & Department of Computer Science, Rutgers University, Piscataway, NJ 08854. Research supported in part by NSF grants CCF-1350572, CCF-1540634, CCF-1909683, BSF grant 2014359, a Sloan research fellowship and the Simons Collaboration on Algorithms and Geometry. Email: shubhangi.saraf@gmail.com.

‡Department of Computer Science, Boston College, Chestnut Hill, MA 02467. Email: ilya.volkovich@bc.edu.

ISSN 1433-8092

1 Introduction

Arithmetic circuits are directed acyclic graphs (DAG) computing multivariate polynomials succinctly, building up from variables using $(+)$ addition and $(\times)$ multiplication operations. *Reconstruction* of arithmetic circuits is the following problem: given black-box (a.k.a oracle/ membership query) access to a polynomial computed by a circuit C of size s from some class of circuits $\mathcal{C}$, give an efficient algorithm (deterministic or randomized) for recovering C or some circuit C' that computes the same polynomial as C. This problem is the algebraic analogue of exact learning in Boolean circuit complexity [Ang88]. If one additionally requires that the output circuit belongs to the same class $\mathcal{C}$ as the input circuit, then it is called *proper learning*.

Reconstruction of arithmetic circuits is an extremely natural problem, but also a really hard problem. Thus in the past few years, much attention has focused on reconstruction algorithms for various interesting subclasses of arithmetic circuits [BBB+00, KS01, KS06, FS12]. In particular, much attention has focused on depth-3 and depth-4 arithmetic circuits [KS09a, GKL12, Sin16, BSV20, Sin20]. Depth-3 and depth-4 circuits have been intensely studied for the problem of proving lower bounds, deterministic polynomial identity testing as well as polynomial reconstruction (which is probably the hardest of the three). Given the depth reduction results of [AV08, Koi10, Tav13, GKKS13], we know that depth-3 and depth-4 arithmetic circuits are very expressive, and good enough reconstruction algorithms (or even lower bounds or polynomial identity testing) for these models would have major implications for general circuits. Thus perhaps not surprisingly, we are quite far from obtaining efficient reconstruction algorithms even for depth-3 circuits.

In this work, we will focus on some interesting subclasses of depth-3 circuits with bounded top fan-in ($\Sigma\Pi\Sigma(k)$ circuits) and give efficient *proper learning* algorithms for them. A setting of particular interest for us (and which motivated much of this work) is when the underlying field is large or infinite (such as $\mathbb{R}$ or $\mathbb{C}$), since in that setting we have even fewer reconstruction algorithms. Though we state many of our results over all fields, for concreteness it will be convenient to imagine the underlying field being $\mathbb{R}$ or $\mathbb{C}$ or $\mathbb{F}_p$.

The subclasses of $\Sigma\Pi\Sigma(k)$ circuits that we study, already capture some very interesting models, and our result for one of these subclasses implies the first efficient polynomial-time algorithm for tensor rank computation and optimal tensor decomposition of *constant-rank tensors*. Before describing the connection to tensors and stating our results, we first give some background on polynomial reconstruction.

There is substantial evidence supporting the hardness of arithmetic circuit reconstruction. Deterministic algorithms for reconstruction are at least as hard as deterministic black-box algorithms for polynomial identity testing, which is equivalent to proving lowering bounds for general arithmetic circuits [KI03, Agr05]. Randomized reconstruction is also believed to be a hard problem and there are a number of results showing hardness of reconstruction under various complexity-theoretic and cryptographic assumptions [Hås90, FK09, KS09c, Shi16]. (For more details see the section on hardness-results in [BSV20]).

Despite reconstruction being a very hard problem, there has been a lot of research focused on efficient reconstruction for restricted classes of arithmetic circuits. Yet, the progress has still been quite slow. Even among the class of constant-depth arithmetic circuits, we only understand reconstruction well for a handful of restricted cases [KS01, KS09a, GKL12, Sin16, BSV20]. If one studies *average case reconstruction* (a model that has received increased attention in recent years) then we know a number of additional results and they hold for richer circuit classes [GKL11, GKQ14, KNST17, KNS18, KS19, GKS20]. However we will not discuss this setting much since the focus of this work will be on the worst case setting.

Before describing the status of what we know about reconstruction for some of the relevant circuit classes, we first define some natural classes of arithmetic circuits that will play an important role in our discussion.

Some Definitions of Relevant Circuit Classes The model of depth-3 arithmetic circuits with top fan-in k, which we refer as $\Sigma\Pi\Sigma(k)$ circuits, has three layers of alternating Σ and Π gates and computes a polynomial of the form

$$C(\bar{x}) = \sum_{i=1}^{k} T_i(\bar{x}) = \sum_{i=1}^{k} \prod_{j=1}^{d_i} l_{ij}(\bar{x})$$

where the $l_{ij}(\bar{x})$-s are linear polynomials.

A multilinear polynomial is a polynomial with individual degree of each variable bounded by 1. We say that a circuit C is multilinear (or syntactically multilinear) if every gate in C computes a multilinear polynomial. Thus, a *multilinear* $\Sigma\Pi\Sigma(k)$ circuit is a $\Sigma\Pi\Sigma(k)$ circuit in which each multiplication gate T_i computes a multilinear polynomial.

A more refined subclass of multilinear polynomial is that of *set-multilinear* polynomials. Let $\sqcup_{j\in[d]}X_j$ be a partition of the set X of input variables. Then a polynomial is set-multilinear under partition $\sqcup_{j\in[d]}X_j$ if each monomial of the polynomial picks up *exactly* one variable from each part in the partition.

A *set-multilinear* $\Sigma\Pi\Sigma(k)$ circuit under partition $\sqcup_{j\in[d]}X_j$ (which we denote as $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuit) is a $\Sigma\Pi\Sigma(k)$ circuit in which each multiplication gate T_i computes a set-multilinear polynomial respecting the partition $\sqcup_{j\in[d]}X_j$. In the Section 1.1 we will discuss this model and its connection to tensor decomposition.

The final subclass of $\Sigma\Pi\Sigma(k)$ circuits that we discuss is the innocuous looking class of sum of power of k linear forms, also referred to as *diagonal* depth-3 circuits with bounded top fan-in ($\Sigma\wedge\Sigma(k)$ circuits). These are a subclass of $\Sigma\Pi\Sigma(k)$ circuits where instead of using multiplication gates, we are just allowed powering gates which raise an input linear polynomial to some power. In Section 1.1 we will discuss this model and its connection to *symmetric* tensor decomposition.

Proper Learning The focus of this work will be on *proper learning* algorithms for subclasses of $\Sigma\Pi\Sigma(k)$ circuits.

Note that in the setting of proper learning, if $\mathcal{C}'$ is a subclass of $\mathcal{C}$, then an efficient proper learning algorithm for $\mathcal{C}$ does not imply an efficient proper learning algorithm for $\mathcal{C}'$. Indeed, as some evidence towards this, note that there are efficient algorithms for proper learning of read-once algebraic branching programs (ROABPs) [BBB+00, KS06, FS12], but we do not know proper learning algorithms for $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}$ circuits and $\Sigma\wedge\Sigma$ circuits (with no bound on the top fan-in), which are both subclasses of ROABPs. In fact, it is known that properly learning $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}$ circuits or $\Sigma\wedge\Sigma$ circuits with an optimal bound for the top fan-in is NP-hard [Hås90, Shi16].

Reconstruction algorithms for $\Sigma\Pi\Sigma(k)$ circuits and for subclasses of $\Sigma\Pi\Sigma(k)$ circuits have been studied in the past a fair bit. The only proper reconstruction algorithms that we are aware of are for the model of multilinear $\Sigma\Pi\Sigma(k)$ circuits by Karnin and Shpilka [KS09a] and for $\Sigma\Pi\Sigma(2)$ circuits by Sinha [Sin16, Sin20]. In the case of $\Sigma\Pi\Sigma(2)$ circuits, the algorithms are proper (i.e. the output is also a $\Sigma\Pi\Sigma(2)$ circuit) only if the "rank" of the linear forms in the underlying circuit is large enough.

All three of these results are highly nontrivial and they introduce several beautiful techniques which give insight into the structure of these models. The Karnin-Shpilka result is in fact more general and gives reconstruction algorithms for $\Sigma\Pi\Sigma(k)$ circuits without the multilinearity constraint, but in this setting the learning algorithms aren't proper (and they do not work over large fields) and we will not discuss it here. For multilinear $\Sigma\Pi\Sigma(k)$ circuits as well, the running time of the Karnin-Shpilka algorithm has a polynomial dependence on the field size $|\mathbb{F}|$. Thus it works only over polynomially-sized finite fields, and in particular it does not work over large or infinite fields (which is the primary focus of this work). We discuss the algorithm from [KS09a] in a little more detail in Section 1.2.

Our goal is to obtain algorithms that work over infinite fields ($\mathbb{R}$, $\mathbb{C}$) with polynomial dependence on the input bit complexity, and that work over finite fields $\mathbb{F}_q$ with $\mathrm{poly}(\log q)$ dependence on the field size. In this setting, the only subclasses of $\Sigma\Pi\Sigma(k)$ circuits for which we know proper learning algorithms is for $\Sigma\Pi\Sigma(2)$ circuits, if the "rank" of the linear forms in the underlying circuit is large enough [Sin16, Sin20]. Both these results use fairly sophisticated tools, and really show why even for the seemingly simple case of $k = 2$, reconstruction can be fairly complex.

Some additional classes of bounded depth circuits for which we do know proper learning algorithms that work over large fields are depth-2 ($\Sigma\Pi$) arithmetic circuits (a.k.a sparse polynomials) which have efficient polynomial-time algorithms [BOT88, KS01], and multilinear depth-4 circuits with top fan-in 2 (multilinear $\Sigma\Pi\Sigma\Pi(2)$ circuits) [GKL12].

1.1 Connection to the Tensor Rank Problem

Tensors, higher dimensional analogues of matrices, are multi-dimensional arrays with entries from some field $\mathbb{F}$. For instance, a 3-dimensional tensor can be written as $\mathcal{T} = (\alpha_{i,j,k}) \in \mathbb{F}^{n_1 \times n_2 \times n_3}$. We will work with general d-dimensional tensors $\mathcal{T} = (\alpha_{j_1,j_2,\ldots,j_d}) \in \mathbb{F}^{n_1 \times \cdots \times n_d}$. The *rank* of a tensor $\mathcal{T}$ can be defined as the smallest r for which $\mathcal{T}$ can be written as a sum of r tensors of rank 1, where a rank-1 tensor is a tensor of the form $v_1 \otimes \cdots \otimes v_d$ with $v_i \in \mathbb{F}^{n_i}$. Here $\otimes$ is the Kronecker (outer) product a.k.a *tensor product*. The expression of $\mathcal{T}$ as a sum of such rank-1 tensors, over the field $\mathbb{F}$ is called $\mathbb{F}$-*tensor decomposition* or just tensor decomposition, for short. The notion of tensor rank/decomposition has become a fundamental tool in different branches of modern science with applications in machine learning, statistics, signal processing, computational complexity, psychometrics, linguistics and chemometrics. We refer the reader to the detailed monograph by Landsberg [Lan12] and the references therein for more details on applications of tensor decomposition.

For a tensor $\mathcal{T} = (\alpha_{j_1,j_2,\ldots,j_d}) \in \mathbb{F}^{n_1 \times \cdots \times n_d}$ consider the following polynomial

$$f_{\mathcal{T}}(X) \triangleq \sum_{(j_1,\ldots,j_d) \in [n_1] \times \cdots \times [n_d]} \alpha_{j_1,j_2,\ldots,j_d} x_{1,j_1} x_{2,j_2} \cdots x_{d,j_d}.$$

Let $C(X) = \sum_{i=1}^{k} \prod_{j=1}^{d} \ell_{i,j}$ be a set-multilinear depth-3 circuit over $\mathbb{F}$ respecting the partition $\sqcup_{j \in [d]} X_j$, and computing $f_{\mathcal{T}}(X)$. Then observe that

$$\mathcal{T} = \sum_{i=1}^{k} \bar{v}(\ell_{i,1}) \otimes \cdots \otimes \bar{v}(\ell_{i,d})$$

where $\bar{v}(\ell_{i,j})$ corresponds to the linear form $\ell_{i,j}$ as an n_j-dimensional vector over $\mathbb{F}$. Indeed, it is easy to see that a tensor $\mathcal{T} = (\alpha_{j_1,j_2,\ldots,j_d}) \in \mathbb{F}^{n_1 \times \cdots \times n_d}$ has rank at most r if and only if $f_{\mathcal{T}}(X)$ can be computed by a $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(r)$ circuit. Therefore, rank of $\mathcal{T}$ is the smallest k for which $f_{\mathcal{T}}(X)$ can be computed by a $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuit.

Consider the following question. **Question 1:** Given as input a 3-dimensional tensor $\mathcal{T} = (\alpha_{i,j,k}) \in \mathbb{F}^{n_1 \times n_2 \times n_3}$, is there an efficient algorithm for computing its tensor rank? This problem is known to be NP-hard in general [Hås90]. Now consider the following variant of the question. **Question 1′:** Given as input a 3-dimensional tensor $\mathcal{T} = (\alpha_{i,j,k}) \in \mathbb{F}^{n_1 \times n_2 \times n_3}$ such that the tensor rank is at most some fixed constant. Does the problem still remain hard, or is the rank efficiently computable? One could also ask these same questions for d-dimensional tensors where d is large. Let $\mathcal{T} = (\alpha_{j_1,j_2,\ldots,j_d}) \in \mathbb{F}^{n_1 \times \cdots \times n_d}$. In such a setting, one might not even be able to efficiently store the entire tensor as an array. However, if the tensor rank is small (say a constant), then there is still a small "implicit" representation of $\mathcal{T}$ a sum of rank one tensors. In this setting, one has black-box access to measurements of $\mathcal{T}$. In particular, given $\bar{\alpha}_i \in \mathbb{F}^{n_i}$ for all $i \in [d]$, the measurement of $\mathcal{T}$ at $(\bar{\alpha_1}, \ldots, \bar{\alpha_d})$ equals $\langle \mathcal{T}, \bar{\alpha}_1 \otimes \cdots \otimes \bar{\alpha}_d \rangle$. The d-dimensional question is strictly harder than the three dimensional question, and again we can ask (***d*-dimensional analog of Question** 1′)- suppose the tensor rank of $\mathcal{T}$ is at most some fixed constant. Is there an efficient algorithm for computing the tensor rank of $\mathcal{T}$?

Observe that each measurement of $\mathcal{T}$ at $(\bar{\alpha_1}, \ldots, \bar{\alpha_d})$ corresponds to a black-box evaluation of the polynomial $f_{\mathcal{T}}$ at $(\bar{\alpha_1}, \ldots, \bar{\alpha}_d)$. Moreover, finding the optimal decomposition of $\mathcal{T}$ as a sum of rank-1 tensors is equivalent to the following: Given black-box access to $f_{\mathcal{T}}$, reconstruct it as a set-multilinear $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}$ circuit with the smallest possible top fan-in.

The three dimensional version was asked as an open question in the work of Schaefer and Stefankovic [SS16]. In a related setting, a version of the d-dimensional variant (efficiently learning an optimal decomposition of a constant-rank tensor by black-box access to the measurements) was also raised in the recent work of Chen and Meka [CM20]. It turns out that the answer to the above question is extremely sensitive to the

underlying field. For instance, if the underlying field is the rationals ($\mathbb{Q}$), then even if the tensor rank is a constant, computing the exact value of the tensor rank over $\mathbb{Q}$ is not known to be decidable (and is, in fact, believed to be undecidable) [Shi16, SS16].

In this paper, we give the first randomized polynomial-time algorithm for computing the tensor rank of a constant-rank, d-dimensional tensor $\mathcal{T}$ [1]. Over the fields $\mathbb{R}$ and $\mathbb{C}$ we also show how to obtain deterministic polynomial time algorithms. Moreover, our algorithm finds the optimal decomposition of $\mathcal{T}$ as a sum of rank-1 tensors. Our algorithm works over fields such as $\mathbb{R}$, large enough finite fields, $\mathbb{C}$, and any other algebraically closed fields. Over other fields, we are only able to compute the tensor rank when we view the entries of the tensor as elements of some extension field.

Theorem 1 (Informal)**.** *Let k be any constant. There exists a randomized polynomial-time algorithm that given black-box access to a polynomial $f \in \mathbb{F}[X]$ computable by a $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuit over $\mathbb{F}$, and the partition $\sqcup X_i$ of the set of variables X, outputs a $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuit computing f. When $\mathbb{F}$ is $\mathbb{R}$ or $\mathbb{C}$ then our algorithm is deterministic.*

This implies a polynomial-time algorithm to compute the optimal tensor decomposition (and hence also the tensor rank) of constant-rank tensors for various fields. The formal version of the result is given in Theorem 1.1

Our proof uses various ingredients such as a variable reduction procedure, and setting up and solving a system of polynomial equations. Another important ingredient used is the *rank bounds* that were developed in the study of polynomial identity testing for $\Sigma\Pi\Sigma(k)$ circuits [DS07, KS07, KS09b, SS09, SS10]. These are structural results for identically zero $\Sigma\Pi\Sigma(k)$ circuits, and essentially show that under some mild conditions, any $\Sigma\Pi\Sigma(k)$ circuit which computes the identically zero polynomial must have its linear forms contained in a "low-dimensional" space. This understanding led to very efficient deterministic polynomial identity testing results for this class, and then eventually were used in efficient reconstruction algorithms for subclasses of $\Sigma\Pi\Sigma(k)$ circuits as well.

Symmetric Tensors: Just as we asked the question of tensor rank computation for general tensors, we can also ask the analogous questions for *symmetric tensors*.

A tensor $\mathcal{T}$ is called symmetric if $X_1 = X_2 = \cdots = X_d$ and we have $\mathcal{T}(i_1, i_2, \ldots, i_d) = \mathcal{T}(j_1, j_2, \ldots, j_d)$ whenever $(i_1, i_2, \ldots, i_d)$ is a permutation of $(j_1, j_2, \ldots, j_d)$. Thus, a symmetric tensor is a higher order generalization of a symmetric matrix. Analogous to tensor rank, *symmetric rank* is obtained when the constituting rank-1 tensors are imposed to be themselves symmetric, that is $\bar{v} \otimes \bar{v} \cdots \otimes \bar{v}$.

Just like in the case of general tensors, computing the symmetric rank reduces to finding the optimal top fan-in of a special class of arithmetic circuits, which is sum of power of linear forms ($\Sigma\wedge\Sigma$) circuits. The class of $\Sigma\wedge\Sigma(k)$ circuits computes polynomials of the form $f = \ell_1^d + \cdots \ell_k^d$ where each ℓ_i is a linear polynomial over the underlying n variables.

Let $C(X) = \sum_{i=1}^{k} \ell_i^d$ be a $\Sigma\wedge\Sigma(k)$ circuit over $\mathbb{F}$ computing $f_{Sym,\mathcal{T}}(X)$ for a symmetric tensor $\mathcal{T} = (\alpha_{j_1,j_2,\ldots,j_d}) \in \mathbb{F}^{n_1 \times \cdots \times n_d}$. Then

$$\mathcal{T} = \sum_{i=1}^{k} \bar{v}(\ell_i) \otimes \cdots \otimes \bar{v}(\ell_i)$$

where $\bar{v}(\ell_i)$ is a n-dimensional vector corresponding to the linear form $\ell_{i,j}$.

Just as in the case of tensor rank, determining the symmetric rank of tensors is also known to be NP-hard [Shi16]. One could still ask if there are efficient algorithms for determining the symmetric rank when the rank is constant. In this paper, we give (what we believe to be) the first randomized polynomial-time algorithm for computing the symmetric tensor rank of a constant-rank d-dimensional symmetric tensor $\mathcal{T}$.

[1] It is possible that the algorithm of Karnin and Shpilka [KS09a] for learning multilinear $\Sigma\Pi\Sigma(k)$ circuits can be adapted to also properly learn set-multilinear $\Sigma\Pi\Sigma(k)$ circuits. The Karnin-Shpilka algorithm has a polynomial dependence on field size $|\mathbb{F}|$. If there algorithm can be adapted then it would give a polynomial-time algorithm over small finite fields. The algorithms in this paper work over infinite fields as well, and that setting was the primary motivation for this work.

Theorem 2 (Informal)**.** *Let k be any constant. Let $\mathbb{F}$ be any field of characteristic 0 or sufficiently large characteristic. There exists a randomized polynomial-time algorithm that given black-box access to a polynomial $f \in \mathbb{F}[X]$ computable by a $\Sigma\wedge\Sigma(k)$ circuit with constant k over $\mathbb{F}$, outputs a $\Sigma\wedge\Sigma(k)$ circuit computing f. When $\mathbb{F}$ is $\mathbb{R}$ or $\mathbb{C}$ then our algorithm is deterministic.*

This implies a polynomial-time algorithm to compute the optimal symmetric tensor decomposition (and hence also the symmetric tensor rank) of constant-rank symmetric tensors over various fields. The formal version of the result is given in Theorem 1.4.

Our proof in this case also uses a variable reduction procedure, and setting up and solving a system of polynomial equations. However the proof is overall way simpler than that for general tensors (and actually fits in about half a page!).

1.2 Multilinear $\Sigma\Pi\Sigma(k)$ circuits

Multilinear $\Sigma\Pi\Sigma(k)$ circuits are a more general class of circuits than $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuits. In the proper learning setting however, a proper learning algorithm for multilinear $\Sigma\Pi\Sigma(k)$ circuits does not imply a proper learning algorithm for $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuits.

In this paper we also study reconstruction algorithms for multilinear $\Sigma\Pi\Sigma(k)$ circuit. Multilinear $\Sigma\Pi\Sigma(k)$ circuits were studied by by Karnin and Shpilka [KS09a] and they give the first polynomial-time algorithm for this class of circuits. However the running time of the Karnin-Shpilka algorithm has a polynomial dependence on the field size $|\mathbb{F}|$. Thus it works only over polynomially sized finite fields, and in particular it does not work over infinite fields [2].

At a very high level, the way the algorithm works in [KS09a] is as follows. It finds a suitable projection of the input circuit where only constantly many variables are kept "alive" and the rest are set to field constants. The new circuit in constantly many variables has only constantly many field elements appearing as coefficients, and hence in time poly($|\mathbb{F}|$) one can efficiently "guess" it by going over all possibilities for what the projected circuit looks like. Once the algorithm hits upon the correct guess of the projected circuit, then it "lifts" the projected circuit to recover the original circuit. The implementation of the lifting procedure is quite clever and uses a very nice clustering procedure. The only place where the prohibitive dependence on the field size comes up is in guessing the projected circuit.

In this work we give the first randomized polynomial-time proper learning algorithm for this model that works over large fields (and in particular infinite fields). Our algorithm works over all fields of characteristic 0 or characteristic greater than d (where d is the degree of the circuit). Over $\mathbb{R}$ and $\mathbb{C}$ we show how to derandomize the above algorithm and to obtain *deterministic* polynomial time algorithms. Several of the ideas in our algorithm are inspired by the algorithm from [KS09a] but we need several new ideas as well.

One similarity we have with [KS09a] is that we also project to constantly many variables and try to learn the projected circuit. Instead of "guessing" or iterating to find the projected circuit, we reduce the problem to solving a suitable system of polynomial equations. The problem is that the projected circuit may not have a unique representation as a multilinear $\Sigma\Pi\Sigma(k)$ circuit, and hence the representation learnt by polynomial system solving might be just some representation (not the original representation) and it might not be liftable. This leads to some subtleties and the rest of the algorithm and how we implement the lift is quite different. We give a more detailed overview in Section 2.3.

Theorem 3 (Informal)**.** *Let k be a constant. Let $\mathbb{F}$ be any field of characteristic 0 or sufficiently large characteristic. There exists a randomized polynomial-time algorithm that given black-box access to a polynomial $f \in \mathbb{F}[X]$ computable by a multilinear $\Sigma\Pi\Sigma(k)$ circuit over $\mathbb{F}$, outputs a multilinear $\Sigma\Pi\Sigma(k)$ circuit computing f. Over $\mathbb{R}$ and $\mathbb{C}$ the algorithms we obtain are in deterministic polynomial time.*

This implies a polynomial-time algorithm for learning multilinear $\Sigma\Pi\Sigma(k)$ circuits over infinite fields. The formal version of the result is given in Theorem 1.6.

[2] The Karnin-Shpilka [KS09a] result is in fact more general and gives reconstruction algorithms for $\Sigma\Pi\Sigma(k)$ circuits without the multilinearity constraint, but in this setting the learning algorithms aren't proper and we will not discuss it.

1.3 Our results

We now state our results. All our algorithms will be randomized algorithms over general fields, and hence algorithms will output the correctly reconstructed circuit with high (say ≥ 0.9) probability. This probability can boosted to $1 - o(1)$ by simply doing independent repetitions. Over $\mathbb{R}$ and $\mathbb{C}$, all our algorithms are deterministic.

Our first main result is a polynomial-time algorithm for proper learning of the class of $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuits.

Theorem 1.1 (Proper learning $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuits)**.** *Given black-box access to a degree d, n variate polynomial $f \in \mathbb{F}[X]$ computable by a $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuit over $\mathbb{F}$, and given the partition $\sqcup X_i$ of the set of variables X, there is a randomized $\mathrm{poly}(d^{k^3}, k^{k^{k^{10}}}, n, c)$ time algorithm for computing a $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuit computing f, where $c = \log q$ if $\mathbb{F} = \mathbb{F}_q$ and c is the maximum bit complexity of any coefficient of f if $\mathbb{F}$ is infinite. When the underlying field $\mathbb{F}$ is $\mathbb{R}$ or $\mathbb{F}_q$ with $q \geq nd \cdot 2^{k+1}$ or algebraically closed, then the output circuit is over $\mathbb{F}$ as well. Otherwise the output circuit is over a degree $\mathrm{poly}(k^{k^{k^{10}}})$ extension of $\mathbb{F}$. Moreover when $\mathbb{F}$ is $\mathbb{R}$ or $\mathbb{C}$, then we show that the above algorithm can be made to run in deterministic time $\mathrm{poly}(d^{k^3}, k^{k^{k^{10}}}, n, c)$.*

We would like to remark that this is the first proper learning algorithm for $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuits, and it works over all fields. We feel this result is particularly interesting in the setting of large or infinite fields such as $\mathbb{R}$ or $\mathbb{C}$, and understanding reconstruction algorithms in that setting was the goal of this work. If we didn't require the learning to be "proper" and were okay with letting the output be a polynomial from a bigger class, then such algorithms were already known (even without the restriction of top fan-in) [BBB+00, KS06].

By the equivalence described in Section 1.1 (see also Lemma 3.29), we obtain the following immediate corollary to Theorem 1.1 which for constant-rank tensors gives us an efficient tensor decomposition algorithm for expressing the input tensor as sum of rank one tensors.

When $\mathcal{T}$ is a d dimensional tensor, as described in Section 1.1, even storing all of $\mathcal{T}$ as an array is too inefficient. However if the rank is small, there is still a small implicit description of $\mathcal{T}$. We consider the setting when we have black-box access to measurements of $\mathcal{T}$ (as described in Section 1.1). This exactly corresponds to having black-box access to the associated polynomial $f_{\mathcal{T}}$.

Corollary 1.2 (Decomposing fixed rank tensors)**.** *Let $\mathcal{T} \in \mathbb{F}^{n_1 \times \cdots \times n_d}$ be a d-dimensional tensor of rank at most k. Let $n = \sum_{i=1}^{d} n_i$. Given black-box access to measurements of $\mathcal{T}$ (equivalently to evaluations of $f_{\mathcal{T}}$), there exists a randomized $\mathrm{poly}(d^{k^3}, k^{k^{k^{10}}}, n, c)$ time algorithm for computing a decomposition of $\mathcal{T}$ as a sum of at most k rank 1 tensors, where $c = \log q$ if $\mathbb{F} = \mathbb{F}_q$ and c is the maximum bit complexity of any coefficient of f if $\mathbb{F}$ is infinite. When the underlying field $\mathbb{F}$ is $\mathbb{R}$ or $\mathbb{F}_q$ with $q \geq nd \cdot 2^{k+1}$ or algebraically closed, then the decomposition is over $\mathbb{F}$ as well. Otherwise the decomposition will be over (a degree $\mathrm{poly}(k^{k^{k^{10}}})$) extension of $\mathbb{F}$. Moreover when $\mathbb{F}$ is $\mathbb{R}$ or $\mathbb{C}$, then we show that the above algorithm can be made to run in deterministic time $\mathrm{poly}(d^{k^3}, k^{k^{k^{10}}}, n, c)$.*

Notice that we can use the above result to obtain an efficient algorithm for computing the exact value of the tensor rank of the input tensor (at least over $\mathbb{R}$, $\mathbb{C}$, large finite fields and other algebraically closed fields). Over other fields we can only compute the tensor rank over an extension field. The way one can compute the tensor rank is as follows: run the above algorithm for all values of k starting from $k = 1$, and the smallest k for which the algorithm successfully outputs a tensor decomposition will be the tensor rank of $\mathcal{T}$. (Note that one can test when the output is successful by a simple randomized polynomial identity test.)

Remark 1.3. *The dependence on k (exponential tower of size 2) is not optimized in the above theorem and corollary and can be improved to a single exponential in k when $\mathbb{F} = \mathbb{C}, \mathbb{R}$, (see Section 3.8 and Section 5 for details). However, the single exponential dependence on k is expected as tensor decomposition is NP-hard in general [Hås90, SS16] and not even known to be computable for $\mathbb{Q}$, thus justifying our need to go to extension fields. See Section 3.9 for more details on hardness of tensor decomposition.*

Note that in the case of constant dimensional tensors (i.e. when one can actually efficiently look at all the entries), we can simulate black-box access to the polynomial $f_{\mathcal{T}}$, given access to the entries of the tensor and vice versa. Thus in the constant dimensional setting our algorithm also gives a way for computing tensor rank and obtaining the optimal tensor decomposition given access to the entries of the tensor. This in particular answers an open question asked by Schaefer and Stefankovic [SS16], who asked as an open question the complexity of computing the tensor-rank when the rank is constant. Our proof in the constant dimensional setting is simpler than that for the setting of growing d. In the setting of d dimensional tensors (for large or growing d) the question of whether one can get improved efficiency when the rank of $\mathcal{T}$ is constant was raised in the work of Chen and Meka [CM20] (in a slightly different context). Our work addresses and resolves this question in the black-box query setting for worst case tensors.

Analogous to the result above for tensor decomposition of general tensors, we also obtain efficient algorithms for optimal symmetric tensor decomposition of constant-rank symmetric tensors. The setting of constant-rank symmetric tensors ends up being much simpler than general tensors, and our proofs for this model are much simpler. This result will follow as a corollary of the next result, which is a randomized polynomial-time algorithm for proper learning of $\Sigma\wedge\Sigma(k)$ circuits.

Theorem 1.4. *(Proper learning $\Sigma\wedge\Sigma(k)$ circuits) Given black-box access to a degree d, n variate polynomial $f \in \mathbb{F}[X]$ computable by a $\Sigma\wedge\Sigma(k)$ circuit over $\mathbb{F}$, such thar $char(\mathbb{F}) > d$ or 0, there is a randomized $\mathrm{poly}\left((dk)^{k^{k^{10}}}, n, c\right)$ time algorithm for computing a $\Sigma\wedge\Sigma(k)$ circuit computing f, where $c = \log q$ if $\mathbb{F} = \mathbb{F}_q$ and c is the maximum bit complexity of any coefficient of f if $\mathbb{F}$ is infinite. When the underlying field $\mathbb{F}$ is $\mathbb{R}$ or $\mathbb{F}_q$ with $q \geq nd2^k$ or algebraically closed, then the output circuit is over $\mathbb{F}$ as well. Otherwise the output circuit is over a degree $\mathrm{poly}((dk)^{k^{k^{10}}})$ extension of $\mathbb{F}$. Moreover when $\mathbb{F}$ is $\mathbb{R}$ or $\mathbb{C}$, then we show that the above algorithm can be made to run in deterministic time $\mathrm{poly}\left((dk)^{k^{k^{10}}}, n, c\right)$.*

By the equivalence described in Section 1.1, we obtain the following immediate corollary to Theorem 1.4 which for constant-rank tensors gives us an efficient symmetric tensor decomposition algorithm for expressing the input tensor as sum of rank one symmetric tensors.

Corollary 1.5 (Decomposing fixed symmetric rank tensors)**.** *Let $\mathcal{T}$ be a symmetric d-dimensional tensor of side length n, with $\mathbb{F}$-entries and symmetric rank at most k, such that $char(\mathbb{F}) > d$ or 0. Given black-box access to $f_{Sym,\mathcal{T}}$, there is a randomized $\mathrm{poly}\left((dk)^{k^{k^{10}}}, n, c\right)$ time algorithm for computing a decomposition of $\mathcal{T}$ as a sum of at most k rank 1 symmetric tensors, where $c = \log q$ if $\mathbb{F} = \mathbb{F}_q$ and c is the maximum bit complexity of any coefficient of f if $\mathbb{F}$ is infinite. When the underlying field $\mathbb{F}$ is $\mathbb{R}$ or $\mathbb{F}_q$ with $q \geq nd2^k$ or algebraically closed, then the decomposition is over $\mathbb{F}$ as well. Otherwise the decomposition will be over (a degree $\mathrm{poly}\left((dk)^{k^{k^{10}}}\right)$ extension of $\mathbb{F}$. Moreover when $\mathbb{F}$ is $\mathbb{R}$ or $\mathbb{C}$, then we show that the above algorithm can be made to run in deterministic time $\mathrm{poly}\left((dk)^{k^{k^{10}}}, n, c\right)$.*

Again, like in the case of general tensor decomposition, Remark 1.3 holds here as well.

We next state our result on proper learning of multilinear $\Sigma\Pi\Sigma(k)$ circuits.

Theorem 1.6 (Proper learning multilinear-$\Sigma\Pi\Sigma(k)$ circuits)**.** *Given black-box access to a degree d, n variate polynomial $f \in \mathbb{F}[X]$ computable by a multilinear $\Sigma\Pi\Sigma(k)$ circuit over $\mathbb{F}$, such that $char(\mathbb{F}) > d$ or 0, there is a randomized $\mathrm{poly}\left(n^{k^{k^{k^{10}}}}, d, k^{k^{k^{10}}}, c\right)$ time algorithm for computing a multilinear $\Sigma\Pi\Sigma(k)$ circuit computing f, where $c = \log q$ if $\mathbb{F} = \mathbb{F}_q$ and c is the maximum bit complexity of any coefficient of f if $\mathbb{F}$ is infinite. When the underlying field $\mathbb{F}$ is $\mathbb{R}$ or $\mathbb{F}_q$ with $q \geq nd \cdot 2^{k+1}$ or algebraically closed, then the output circuit is over $\mathbb{F}$ as well. Otherwise the output circuit is over a degree $\mathrm{poly}\left((k)^{k^{k^{k^{10}}}}\right)$ extension of $\mathbb{F}$. Moreover when $\mathbb{F}$ is $\mathbb{R}$ or $\mathbb{C}$, then we show that the above algorithm can be made to run in deterministic time $\mathrm{poly}\left(n^{k^{k^{k^{10}}}}, d, k^{k^{k^{10}}}, c\right)$.*

This is the first efficient proper learning algorithm for multilinear-$\Sigma\Pi\Sigma(k)$ circuits that works over large fields, and in particular infinite fields such as $\mathbb{R}$ and $\mathbb{C}$. Even here, the dependence on k (exponential tower of size 3) is not optimized in the above theorem and can be improved to a tower of size 2 in k when $\mathbb{F} = \mathbb{C}, \mathbb{R}$, (see Section 6 for details).

Deterministic vs Randomized Reconstruction Algorithms: The algorithms we give in this paper are randomized over general fields and deterministic over $\mathbb{R}$ and $\mathbb{C}$. Indeed, derandomizing them in general, will be highly nontrivial for the following reason. In the reconstruction problem for all three subclasses of $\Sigma\Pi\Sigma(k)$ circuits being studied, we can embed within them the problem of solving a system of polynomial equations. (See Theorem 3.37 and the discussion in Section 3.9.) The only efficient algorithms we know for solving systems of polynomial equations over large finite fields (i.e with running time polynomial in $\log q$ for a field $\mathbb{F}_q$) are randomized and it is a very interesting open question to derandomize them. A derandomized solution to our reconstruction algorithms over large finite fields would have very interesting algorithmic implications for polynomial system solving, see [AM94, Problem 15].

Interestingly, the large characteristic case is the only case when low-variate polynomial system solving is hard to derandomize. That is, if the underlying field is not a finite field with large characteristic, then there do exist efficient deterministic algorithms for low-variate polynomial system solving. See Section 3.8.1 for details. Also, this turns out to be the only bottleneck for derandomizing our learning/decomposition algorithms. That is, if the underlying field is not a finite field with large characteristic, then the algorithms underlying Theorems 1.1, 1.4, 1.6 can be derandomized efficiently. Though we do not mention this explicitly, it is easy to see that, when $\mathbb{F} = \mathbb{F}_{p^t}$ then the algorithms mentioned in Theorems 1.1, 1.4, 1.6 can be made deterministic with an additional polynomial in p (characteristic) dependence in time complexity. See derandomization remarks in respective sections for details.

When we present our proofs, for simplicity we will first present the randomized algorithms and then later point out the changes that need to be made in order to derandomize them.

1.4 Related/Previous Work

Reconstruction of $\Sigma\Pi\Sigma(k)$ circuits has received a fair amount of attention. The case of $k = 1$ is resolved by the black-box factoring algorithm of Kaltofen and Trager [KT90]. The case of $k = 2$ is already highly nontrivial and very interesting and thus needed quite a few new ideas. This case was first studied by Shpilka [Shp09], who designed a reconstruction algorithm for $k = 2$ which was later improved by Karnin and Shpilka [KS09a] who gave efficient reconstruction algorithms for $(\Sigma\Pi\Sigma(k))$ circuits for any constant top fan-in k. When the input is an n-variate, degree d polynomial computed by a size s circuit, both algorithms run in time quasipoly$(n, d, |\mathbb{F}|, s)$. The algorithms are not 'proper learning' algorithms, and the output is from a larger class of "generalized" depth-3 circuit. Moreover given the dependence of the running time on the field size, these algorithms aren't efficient over large/infinite fields.

Over fields of characteristic 0, the only efficient reconstruction algorithm we know for $\Sigma\Pi\Sigma(k)$ circuits is the randomized algorithm by [Sin16] which works for $k = 2$, and uses lots of new ideas such as quantitative/robust Sylvester-Gallai theorems for high dimensional points. Very recently, in [Sin20], Sinha studied the case of $k = 2$ for finite fields and gave the first algorithm in this setting with poly log dependence in field size. These algorithms are mostly proper, but not always. When the rank of the linear forms in the input polynomial is not high dimensional, then the output circuit might not be a $\Sigma\Pi\Sigma(2)$ circuit.

When the input is a *multilinear* $\Sigma\Pi\Sigma(k)$ circuit, the works of Shpilka [Shp09] and Karnin-Shpilka [KS09a] give polynomial-time *proper learning* algorithms. The dependence on the field size is still poly$(|\mathbb{F}|)$, and hence these algorithms do not work over large/infinite fields. Inspired by the work of Karnin and Shpilka, in [BSV20] similar results were obtained for multilinear depth-4 circuits with bounded top fan-in ($\Sigma\Pi\Sigma\Pi(k)$ circuits). The running time is however still at least poly$(|\mathbb{F}|)$, and hence it does not work over large/infinite fields. When the top fan-in is 2, i.e. for $\Sigma\Pi\Sigma\Pi(2)$ circuits, we do know such efficient polynomial-time reconstruction algorithms by the work of Gupta, Kayal and Lokam [GKL12].

Other Results: The class of circuits for which we understand reconstruction really well is the class of depth-2 ($\Sigma\Pi$) arithmetic circuits (a.k.a sparse polynomials). We can properly learn sparse polynomials in *deterministic* poly(s, n, d) time over any field [BOT88, KS01]. Another class for which we understand reconstruction *reasonably* well is the class of read-once oblivious branching programs (ROABPs). Klivans and Shpilka [KS06] gave a randomized reconstruction (proper learning) algorithm for ran in time poly(n, d, s). This was later derandomized in [FS12] with time complexity quasipoly(n, d, s). For depth-3 circuits, reconstruction algorithms for various other restricted classes have been studied. For instance, for set-multilinear depth-3 circuits [BBB+00, KS06] gave a randomized poly(n,d,s) (improper) learning algorithm which outputs an ROABP.

Recently, there has been a flurry of activity in average case learning algorithms for various arithmetic circuit classes [GKL11, GKQ14, KNST17, KNS18, KS19, GKS20]. These results can be thought of as worst case reconstruction, given some non-degeneracy condition holds for some implicit polynomials (which are usually computed by intermediate gates). Interestingly, these results fall under the umbrella of learning from natural lower bounds which is an exciting area of research in arithmetic as well as Boolean circuit complexity [CIKK16, KS19].

2 Proof Overview

We have three main results in the paper:(1) reconstruction of $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuits (equivalent to low rank tensor decomposition), (2) reconstruction of $\Sigma\wedge\Sigma(k)$ circuits (equivalent to low rank symmetric tensor decomposition), and (3) reconstruction of multilinear $\Sigma\Pi\Sigma(k)$ circuits.

Our algorithms are randomized over general fields and we show how to derandomize then over $\mathbb{R}$ and $\mathbb{C}$. For simplicity, in the proof overview we will only discuss the randomized algorithms. Later in the paper when we give the formal proof we will show how to derandomize the algorithms.

A common theme in the proof of each of these results is that all proofs involve a *variable reduction procedure* and setting up and *solving a suitable system of polynomial equations*, where a solution to the system gives some important information about the circuit being reconstructed. In the case of reconstruction for $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuits and multilinear $\Sigma\Pi\Sigma(k)$ circuits, the proofs are considerably more involved and also use "rank bound" techniques that give structural information about $\Sigma\Pi\Sigma(k)$ circuits that are identically 0.

For simplicity, we start with a proof overview of the result that was (in hindsight) quite easy to prove, which is coming up with an efficient reconstruction algorithm for $\Sigma\wedge\Sigma(k)$ circuits.

2.1 Reconstruction of $\Sigma\wedge\Sigma(k)$ circuits

Let f be a polynomial which has a $\Sigma\wedge\Sigma(k)$ representation, and let

$$C_f \equiv \sum_{i=1}^{k} (a_{i,1}x_1 + a_{i,2}x_2 + a_{i,3}x_3 + \ldots + a_{i,n}x_n)^d$$

be the $\Sigma\wedge\Sigma(k)$ circuit computing f.

An important observation is that if f can be represented by a $\Sigma\wedge\Sigma(k)$ circuit, then f has only k "essential variables". In particular one can apply an invertible linear transformation to the variables of f so that the transformed f only depends on k variables.

What is nice is that such a linear transformation can actually be computed without actually looking at C_f and its linear forms, but only with black-box access to f. This follows from result of Kayal [Kay11], and which built upon a result by Carlini [Car06]. (The original result by Kayal was not stated or used in the black-box setting, but it is easy to see that the proof an be adapted to black-box setting as well.) Let $g_A(\bar{x}) = f(A \cdot \bar{x})$, where $g_A(\bar{x})$ depends only on k variables. Since the algorithm can compute A, hence given black-box access to f, it can efficiently simulate black-box access to g_A. Moreover, observe that g_A also has a $\Sigma\wedge\Sigma(k)$ representation. Thus if we can learn a $\Sigma\wedge\Sigma(k)$ representation of g_A, then by simply applying the inverse linear transform, one can recover a $\Sigma\wedge\Sigma(k)$ representation of f.

Thus the new goal is to learn a $\Sigma\wedge\Sigma(k)$ representation of g_A given black-box access to it. We will do this be reducing the problem of learning the $\Sigma\wedge\Sigma(k)$ representation of g_A to solving a suitable system of polynomial equations. Recall that g_A only depends on k variables. Thus the monomial representation of g_A only has $\binom{k+d}{d}$ monomials. Since k is small, this quantity is not too big, and one can invoke black-box reconstruction algorithms for sparse polynomials [BOT88, KS01] to learn g_A as a sum of monomials. Let $g_A = \sum_{\bar{e}} c_{\bar{e}} \cdot \bar{x}^{\bar{e}}$ be the monomial representation of g_A.

Let

$$C_{g_A} \equiv \sum_{i=1}^{k} (b_{i,1}x_1 + b_{i,2}x_2 + b_{i,3}x_3 + \ldots + b_{i,k}x_k)^d$$

be a $\Sigma\wedge\Sigma(k)$ representation of g_A.

Then notice that

$$\sum_{i=1}^{k} (b_{i,1}x_1 + b_{i,2}x_2 + b_{i,3}x_3 + \ldots + b_{i,k}x_k)^d = \sum_{\bar{e}} c_{\bar{e}} \cdot \bar{x}^{\bar{e}}.$$

Now for each monomial $\bar{x}^{\bar{e}}$ that appears in g_A, we can compare the coefficient of $\bar{x}^{\bar{e}}$ on both sides of the above expression to get a polynomial equation in the variables $b_{i,j}$. Doing this for all monomials gives us a system of at most $\binom{k+d}{d}$ polynomial equations in k^2 variables, with $b_{i,j}$ as the unknown variables. Observe that any solution to the system of equations would give a $\Sigma\wedge\Sigma(k)$ representation of g_A an vice versa. By Theorem 3.36, this system can be solved in polynomial time if k is a constant.

2.2 Reconstruction of $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuits

We now show how to efficiently reconstruct $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuits. Again, variable reduction and setting up and solving polynomial systems of equations play an important role, but several other ingredients (such as rank bound techniques) also go into the proof and the proof is more involved.

We are given as input black-box access to a degree d, n variate polynomial $f \in \mathbb{F}[X]$ computable by a $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuit over $\mathbb{F}$, and we are also given the partition $\sqcup X_i$ of the set of variables X. Let $C_f \equiv \sum_{i=1}^{k} \prod_{j=1}^{d} \ell_{i,j}$, be a $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ representation of f, where each $\ell_{i,j}$ is a linear polynomial in X_j variables.

2.2.1 Variable reduction:

As a first step, we show how to reduce the number of variables in each part to at most k. Here we cannot directly invoke the result by Kayal [Kay11] and Carlini [Car06] for the following reasons. The total number of essential variables is $k \times d$ which is quite large. Though the number of essential variables in every part is at most k, there seems to be no straightforward way to apply the result separately to each part[3]. Even if kd was small, after applying the linear transformation given by the Carlini-Kayal result, the new circuit might not be set-multilinear, and we need to crucially maintain set-multilinearity in order for the other steps of the algorithm to be carried out.

Instead, we use the structural properties of set-multilinear circuits to come up with a a different black-box algorithm for performing the variable reduction. We essentially come up with d different invertible linear transformations, one for each set of variables in the partition, that reduces the variables in each set to at most k. In Section 5.1 we elaborate more on how we find these transformations using some properties of the underlying class of circuits. After this step is performed, one can essentially assume that the input circuit is such that each set of the partition has at most k variables.

2.2.2 Reconstructing low degree ($d \leq 2k^2$) $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuits:

Once we have the variable reduction established, we proceed along the same lines as the algorithm for reconstructing $\Sigma\wedge\Sigma(k)$ circuits. Since the degree is small, the number of monomials appearing in f is small,

[3]Since the linear maps might then end up being over the field of rational functions in the remaining variables.

and the total number of variables appearing in f is small. (Unlike the symmetric case where the number of monomials was small even for high degree circuits). One can invoke black-box reconstruction algorithms for sparse polynomials [KS01, BOT88] to learn f as a sum of monomials. Then, similar to the $\Sigma \wedge \Sigma(k)$ case, we set up a system of polynomial equations in poly(k) variables such that every solution to the system corresponds to a $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ representation of f. For more details, see Section 5.2.

2.2.3 Reconstructing high degree ($d > 2k^2$) $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuits:

The high level plan for reconstructing general high degree $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuits is to use induction on k. When $k = 1$, then the algorithm just invokes a black-box factoring algorithm such as [KT90]. Now assume $k \geq 2$.

Our first step will be to just learn any one linear form appearing in C_f. (Actually as a first step it will be convenient to learn two distinct linear forms such that each multiplication gate contains at most one of them.) In the next step we will use that linear form to learn most of the linear forms of C_f. In the final step we will try to learn all the linear forms and obtain a full $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ representation of f.

Learning one (or two) linear forms appearing in C_f: The algorithm chooses k^2 sets of variables in the partition $X = \sqcup X_i$ to keep "alive" and sets the variables in the remaining sets to random values. Let the resulting restricted polynomial be f_R and the resulting constant degree $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuit be C_{f_R}.

Now, we already know reconstruction algorithms (from the previous case) for low degree $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuits which we could invoke. If we could learn C_{f_R}, then in particular we would have learnt several linear forms of C_f. However note that all we have is black-box access to f_R, which might not have a unique $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuit representation. In fact it might have exponentially many $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuit representations, and our reconstruction algorithm would learn one of these representations. Thus it is possible that we do not learn C_{f_R}, but some other $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuit representation of f_R, call it C'_{f_R} . Now a priori it may seem that the linear forms in C'_{f_R} might not have anything in common with the linear forms of C_{f_R} or C_f. However using rank bound arguments that have been used extensively in the past to analyze identically 0 $\Sigma\Pi\Sigma(k)$ circuits (for polynomial identity testing and polynomial reconstruction), one can show two distinct $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ representations of the same polynomial must indeed have many linear forms in common (as long as the degree is large enough, which it is in our case). Thus we get that C_{f_R} and C'_{f_R} (which we learnt) must have many linear forms in common. Though we may not know exactly which linear form of C'_{f_R} also appears in C_f, we can come up with a small list of candidate options and then iterate over these options. Any wrong candidate will not lead to a successful output of the final algorithm and we will be able to detect it by a later testing phase. Thus we can effectively assume we know a linear form in C_f. In fact if we do things more carefully we can ensure that we know two linear forms ℓ_1 and ℓ_2 appearing in C_f such that they are supported on the same subset of variables.

Learning most of the linear forms from each multiplication gate of C_f: Once we learn ℓ_1 and ℓ_2 appearing in C_f, we try to learn more linear forms as follows. (We don't need f_R any more or C'_{f_R})

The algorithm applies a suitable random setting of the variables of ℓ_1 in the polynomial f, that makes ℓ_1 evaluate to 0, and results in a circuit with $< k$ multiplication gates. Call the restricted polynomial f_{R_1} and let $C_{f_{R_1}}$ be the restricted version of C_f. By the inductive hypothesis, we can learn a $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k-1)$ representation of f_{R_1}. Call this $C'_{f_{R_1}}$. If we could actually learn the representation $C_{f_{R_1}}$ then we would have learnt most of the linear forms in all the multiplication gates of C_f that did not get set to zero under the restriction. However we can only learn some other representation, which we called $C'_{f_{R_1}}$. Using rank bound arguments, we will however still be able to argue that $C'_{f_{R_1}}$ and $C_{f_{R_1}}$ have a lot in common. In fact we show that each multiplication gate of $C_{f_{R_1}}$ overlaps almost entirely (in all but k linear forms) with some multiplication gate of $C'_{f_{R_1}}$. Repeating this procedure for the other linear form ℓ_2 as well gives us another restricted circuit $C_{f_{R_2}}$ and the version of it that is learnt which is $C'_{f_{R_2}}$. It is now easy to see that each multiplication gate of C_f overlaps almost entirely (in all but k linear forms) with some multiplication gate of $C'_{f_{R_1}}$ or $C'_{f_{R_1}}$.

Once we have this, by iterating over all ways of matching up the multiplication gates and choices of overlap, we can make generate a polynomial sized list of k-tuples $(G_1, G_2, \ldots G_k)$ which has the following property. One of the k-tuples $(G_1, G_2, \ldots G_k)$ from the list will have the property that $f = G_1H_1 + \cdots G_kH_k$ and $G_iH_i = T_i$ where T_i was one of the multiplication gates in the original $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ representation of f, C_f. Each G_i has degree $d - k^2$ and hence each H_i has degree k^2. By a little bit of more effort we can also ensure that all the H_i depend on the same sets of the underlying variable partition. The final algorithm will go over all possible k-tuples $(G_1, G_2, \ldots G_k)$ from the list in order to find the correct one. All the wrong ones will not lead to a successful reconstruction, and will get eliminated by a later testing phase.

Learning the full $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ representation of f: We now assume that we have learnt k polynomials $G_1, G_2, \ldots, G_k$ such that $f = G_1H_1 + \cdots G_kH_k$. $G_iH_i = T_i$ where T_i was one of the multiplication gates in the original $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ representation of f. Each H_i is a polynomial in k^3 variables of degree at most k^2 (since after variable reduction each part had at most k variables) and all the H_i depend on the same sets of the underlying variable partition. We need to now learn the H_i, or even some variation of them which will eventually lead to a full $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ representation of f.

We demonstrate how we do this with some simple examples. As a simple case, suppose that the G_i are linearly independent polynomials. By substituting random values into the variables of the G_i, we obtain black-box access to a random linear combination of $H_1, \ldots H_k$. Call this linear combination P_1. From black-box access to P_1, we can actually obtain the monomial representation of P_1 using black-box interpolation for sparse polynomials. We can repeat this process k times to get k different random linear combinations of $H_1, \ldots H_k$. The linear independence of $G_1, G_2, \ldots G_k$ implies that these random linear combinations will be linearly independent with high probability (see Lemma 3.16). Since we know the G_i, we actually know to coefficients of the random linear combinations. Thus once we learn these combinations, we can invert the transformation and actually get black-box access to each H_i individually. Once we have black-box access to each H_i, we can factorize them in a black-box way and hence recover the full underlying circuit.

Here is a slightly more general case. Imagine that $k = 3$, G_1 and G_2 are independent, but $G_3 = G_1 + G_2$. Since we actually know the G_is, we can learn their linear dependency structure (for instance by taking enough random evaluations of them and learning the linear dependence structure of the evaluations, see Lemma 3.17). Then,

$$C_f = G_1H_1 + G_2H_2 + (G_1 + G_2)H_3 = G_1(H_1 + H_3) + G_2(H_2 + H_3)$$

Let $H_1 + H_3 = K_1$ and $H_2 + H_3 = K_2$. Now just as in the simple case when all the G_is were independent, we can again learn the monomial representation of two distinct random linear combinations of K_1 and K_2, and then use this to recover the monomial representations of K_1 and K_2. What remains is to find a representation of K_1 which looks like $H_1 + H_3$ and a representation of K_2 which looks like $H_2 + H_3$. Individually, each looks like a case of finding a $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(2)$ representation for low degree polynomials, but these two $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(2)$ representations are entangled since they must share a multiplication gate. However we can set up one big system of polynomial equations for solving both these reconstruction problems at the same time that takes into account the shared multiplication gate.

This more general case that we just described contains most of the ideas for the fully general case. For more details, refer to Section 5.5.

2.3 Reconstruction of multilinear $\Sigma\Pi\Sigma(k)$ circuits

We now give a proof overview and describe our algorithm for efficiently learning multilinear $\Sigma\Pi\Sigma(k)$ circuits. The main goal of this result is to find a procedure which also works over large and infinite fields.

Variable reduction and setting up and solving polynomial systems of equations again play an important role, especially for the case of low degree multilinear $\Sigma\Pi\Sigma(k)$ circuits. However the implementation of this technique and how to set up and solve the system of equations is more subtle. For general high degree multilinear $\Sigma\Pi\Sigma(k)$ circuits, we need several other tools such as a clustering procedure (inspired by the work

of [KS09a], rank bounds, the notion of rank preserving subspaces, black-box factoring algorithms and an error correcting procedure.

2.3.1 Reconstruction of low degree multilinear $\Sigma\Pi\Sigma(k)$ circuits

Think of k (the top fan-in) and d (the degree) to be constants, and the number of variables, n, to be growing. Let $f \in \mathbb{F}[x_1, x_2, \ldots, x_n]$ be a polynomial computed by a degree d, multilinear $\Sigma\Pi\Sigma(k)$ circuit C of the form

$$\sum_{i=1}^{k} T_i(\bar{x}) = \sum_{i=1}^{k} \prod_{j=1}^{d_i} \ell_{i,j}(\bar{x}) \tag{1}$$

where for each fixed i, the different $\ell_{i,j}$ are supported on disjoint variables.

Let m be the number of essential variables in f. Since there at most kd linear forms appearing in C, it is easy to see that the number of essential variables in f, i.e. m, is at most kd.

We now apply a variable reduction procedure, and for this we invoke the result by Kayal[Kay11] and Carlini [Car06] (Lemma 3.23) to efficiently compute an invertible linear transformation $A \in \mathbb{F}^{n \times n}$ such that $f(A \cdot \bar{x})$ only depends on the first m variables.

Let $g(\bar{x}) = f(A \cdot \bar{x})$. Observe that given black-box access to f, one can easily simulate black-box access to g. Also since $g(A^{-1} \cdot \bar{x}) = f(\bar{x})$, any algorithm that can efficiently learn g can also efficiently learn f in the following way. For each $i \in [n]$, suppose that R_i denote the ith row of A^{-1}. Then in the ith input to g we simply input the linear polynomial $L_i = \langle R_i, \bar{x} \rangle$, which is the inner product of R_i and the vector $\bar{x}$ of formal input variables. Since g only depends on the first m variables, we only really need to do this operation for $i \in [m]$.

Since f is computed by a degree d multilinear $\Sigma\Pi\Sigma(k)$ circuit, hence $g(X) = f(A \cdot \bar{x})$ also has a natural degree d $\Sigma\Pi\Sigma(k)$ circuit representation, where the linear forms of that representation are obtained by applying the transformation A to corresponding linear forms of C. Let us call this circuit C_g. Notice that C_g *may not be multilinear*. However, if were somehow able to learn the precise circuit C_g, then by substituting each variable x_i to L_i then we would recover the circuit C which is indeed multilinear.

Thus our goal is now the following. We have black-box access to g which only depends on m variables. We would like to devise an algorithm for reconstructing C_g. Now here is a slight issue. C_g is a *particular* degree d $\Sigma\Pi\Sigma(k)$ representation of g. It has the nice property that when we plug in $x_i = L_i$ (for all $i \in [m]$) in this representation, then we recover a multilinear $\Sigma\Pi\Sigma(k)$ representation of f. Let us call the new circuit obtained by plugging in $x_i = L_i$ for each i, the "lift" of C_g. Observe that g might have multiple (perhaps exponentially many) representations as a degree d $\Sigma\Pi\Sigma(k)$ circuit. If given black-box access to g, the reconstruction algorithm finds some other degree d $\Sigma\Pi\Sigma(k)$ representation of g, call it C'_g, then there is no guarantee that when we plug in $x_i = L_i$ in this representation, then we recover a multilinear $\Sigma\Pi\Sigma(k)$ representation of f. In other words, the lift of C'_g in general *may not be multilinear*.

Although in our algorithm we will not actually be able to guarantee that we learn precisely C_g, however the existence of C_g tells us that *there exists* a $\Sigma\Pi\Sigma(k)$ representation of g whose lift is a multilinear $\Sigma\Pi\Sigma(k)$ circuit. We will use this existence to actually find a suitable $\Sigma\Pi\Sigma(k)$ representation of g whose lift is multilinear.

In order to learn a degree d $\Sigma\Pi\Sigma(k)$ representation of g we will set up a system of polynomial equations such that any solution to it will give as a degree d $\Sigma\Pi\Sigma(k)$ representation of g. (We do this in a very similar manner to how we did it for $\Sigma \wedge \Sigma(k)$ circuits and $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuits.) We then show how to impose several additional polynomial constraints to this system that will further ensure that whatever $\Sigma\Pi\Sigma(k)$ representation is learnt will be such that its lift will be a multilinear $\Sigma\Pi\Sigma(k)$ circuit. The details of how we implement this can be found in Lemma 6.4.

2.3.2 Reconstructing general (high degree) multilinear $\Sigma\Pi\Sigma(k)$ circuits

We now describe our algorithm for reconstructing general multilinear $\Sigma\Pi\Sigma(k)$ circuits. What we describe here is a bit of a simplification and it avoids some technical issues, but we hope that it provides a high level picture of the algorithm.

Clustering of gates: We use a very nice and elegant clustering procedure devised in the work of Karnin and Shpilka [KS11] (which they used for reconstructing $\Sigma\Pi\Sigma(k)$ circuits over small fields). We will not describe the algorithm here, but describe some nice properties that the clustering satisfies. Given as input $C = \sum_i T_i$ where the T_i are the multiplication gates of a degree d multilinear $\Sigma\Pi\Sigma(k)$ circuit C, the clustering algorithm looks at the T_i and outputs a partition of the the k multiplication gates into a set of clusters $C_1, C_2, \ldots C_r$ (for some $r \in [k]$). Each cluster C_i is some subset of the multiplication gates of C, and has the property that any two multiplication gates in a cluster are very "close" to each other. Suppose that $C_i = \{T_{i_1}, T_{i_2}, T_{i_3}\}$. Then consider the associated circuit $C'_i = T_{i_1} + T_{i_2} + T_{i_3}$. The closeness of every two of the multiplication gates will imply that one can write C'_i as

$$C'_i = T_{i_1} + T_{i_2} + T_{i_3} = \gcd(T_{i_1}, T_{i_2}, T_{i_3}) \times \left(T'_{i_1} + T'_{i_2} + T'_{i_3}\right)$$

where $T'_{i_1} + T'_{i_2} + T'_{i_3}$ is a *low degree* multilinear $\Sigma\Pi\Sigma(3)$ circuit. Now notice that we don't know what C is (that is what we are trying to learn) and hence we cannot apply any clustering procedure to it. However this clustering exists, and it is canonical. We only have black-box access to the original circuit C. Suppose that we could somehow obtain black-box access to each of the clusters (or rather to the circuits corresponding to the clusters). We would then actually be done! Here is why. Suppose we had black-box access to C'_i, then we would first apply a black-box factoring algorithm (such as that given by [KT90]) to compute all the linear factors of C'_i (thus we would obtain $\gcd(T_{i_1}, T_{i_2}, T_{i_3})$) and divide them out. We would then be left with black-box access to $T'_{i_1} + T'_{i_2} + T'_{i_3}$ is a *low degree* multilinear $\Sigma\Pi\Sigma(3)$ circuit. But we already saw how to reconstruct low degree multilinear $\Sigma\Pi\Sigma(3)$ circuits! By multiplying it with its linear factors, we we would be able to recover a multilinear $\Sigma\Pi\Sigma(3)$ circuit for C_i. We would repeat this procedure for each cluster and then put it all together to obtain a multilinear $\Sigma\Pi\Sigma(k)$ representation for C.

Thus the goal from now on will be to somehow obtain black-box access to the clusters. The clustering output by the clustering algorithm also has some additional nice properties. It is a "robust" clustering, that is, if two multiplication gates got assigned to different clusters, then they are quite "far" from each other (in some well defined sense). This nice property ends up implying the following. We start with the circuit C in n variables. Then there is some constant number (about k^k) of variables one can keep "alive" (call these the $\bar{y}$ variables) such that if we set the remaining variables (call these the $\bar{z}$ variables) to random values ($\bar{z} = \bar{\alpha}$), then the new restricted circuit $C|_{\bar{z}=\bar{\alpha}}$ has the following property. Suppose we applied the clustering algorithm to $C|_{\bar{z}=\bar{\alpha}}$, then the clusters obtained would exactly match up with the clusters output by the clustering algorithm applied to the circuit C, and each cluster of $C|_{\bar{z}=\bar{\alpha}}$ would be obtained by the same restriction procedure being applied to the corresponding cluster of C.

Obtaining access to evaluations of the clusters at random inputs: Notice that though we do not know what C is, we can know what $C|_{\bar{z}=\bar{\alpha}}$ is. This is because $C|_{\bar{z}=\bar{\alpha}}$ has only about k^k variables and hence is a low degree multilinear $\Sigma\Pi\Sigma(k)$ circuit. Hence we can reconstruct it. We have to be a bit careful here since our reconstruction algorithm might not output the precise circuit $C|_{\bar{z}=\bar{\alpha}}$ but some other multilinear $\Sigma\Pi\Sigma(k)$ circuit representation of the same polynomial, call it $C'|_{\bar{z}=\bar{\alpha}}$. However the clustering procedure turns out to be robust enough that the clusters of $C|_{\bar{z}=\bar{\alpha}}$ and the clusters of $C'|_{\bar{z}=\bar{\alpha}}$ match up to compute the same polynomials. Hence we can essentially assume that we know what $C|_{\bar{z}=\bar{\alpha}}$ is and hence we can cluster its gates as well. By the properties of clustering, the clusters of $C|_{\bar{z}=\bar{\alpha}}$ match up with the clusters of C (after we set the $\bar{z} = \bar{\alpha}$). Thus though we do not as yet have black-box access to the clusters of C, we can indeed recover what the clusters look like after setting $\bar{z} = \bar{\alpha}$. Thus if $C'_1, C'_2, \ldots C'_r$ are circuits corresponding to the clusters of C, then we can recover their restrictions to $\bar{z} = \bar{\alpha}$. Notice that α was any random sample from $\mathbb{F}^m$, where m is the number of Z variables. Thus we can essentially recover black-box evaluations of the clusters at *randomly chosen inputs*. If we could do the same for the Z variables being set to any arbitrary adversarially chosen $\beta \in \mathbb{F}^m$ then we would be done.

There is one issue we have swept under the rug, which is the following. The clusters of $C|_{\bar{z}=\bar{\alpha}}$ match up with the clusters of C, but *we don't know what this matching is*. In particular, we might be able to learn $C|_{\bar{z}=\bar{\alpha}}$ as well as $C|_{\bar{z}=\bar{\alpha}'}$ for two distinct $\bar{\alpha}, \bar{\alpha}' \in \mathbb{F}^m$, and we might be able to cluster both of them, and these clusters correspond to the clusters of C, but since we don't know the correspondence we cannot really say

that we know the value of $C'_i|\bar{z} = \bar{\alpha}$ as well as $C'_i|_{\bar{z}=\bar{\alpha}'}$ for the same C'_i. We will refer to this as "ambiguity issue".

Obtaining the corresponding between two clusterings: We now address the ambiguity issue. Suppose we know what $C'_i|_{\bar{z}=\bar{\alpha}}$ looks like. We would like to be able to compute $C'_i|_{\bar{z}=\bar{\alpha}'}$ for any other randomly chosen $\alpha' \in \mathbb{F}^m$. Note that we can reconstruct $C|\bar{z} = \bar{\alpha}'$ and cluster it and that would give us the set $\{C'_1|_{\bar{z}=\bar{\alpha}'}, C'_1|_{\bar{z}=\bar{\alpha}'}, \ldots C'_r|_{\bar{z}=\bar{\alpha}'}\}$, but we may not know which element of the set corresponds to $C'_i|_{\bar{z}=\bar{\alpha}'}$. In order to do this identification, we first show how to do this when $\bar{\alpha}$ and $\bar{\alpha}'$ differ in only one coordinate, and then we use a hybrid argument to stitch it together for general $\bar{\alpha}$ and $\bar{\alpha}'$ (by considering a sequence of different αs going from α to α' and with consecutive elements differing in one coordinate). When $\bar{\alpha}$ and $\bar{\alpha}'$ differ in only one coordinate, we observe that $C'_i|_{\bar{z}=\bar{\alpha}}$ and $C'_i|_{\bar{z}=\bar{\alpha}'}$ are very similar or very "near each other" in a suitably defined metric. Then using the robustness property of the clustering we show that the identification of $C'_i|_{\bar{z}=\bar{\alpha}'}$ can be done.

From evaluations at random points to evaluations at worst case points: Let C'_i be the circuit corresponding to cluster C_i. Let us assume we know how to compute $C'_i|_{\bar{z}=\bar{\alpha}}$ for any randomly chosen $\alpha \in \mathbb{F}^m$. Now let $\bar{\beta}$ be some arbitrary point in $\mathbb{F}^m$. We would like to compute $C'_i|_{\bar{z}=\bar{\beta}}$. We use Reed-Solomon decoding for this. We consider the line $t \cdot \bar{\alpha} + (1-t) \cdot \bar{\beta}$ passing through $\bar{\alpha}$ and $\bar{\beta}$ in $\mathbb{F}^m$. In order to learn $C'_i|_{\bar{z}=\bar{\beta}}$, we will learn the restriction of C'_i to the full line, which is a polynomial in the Y variables and the additional t variable. Then setting $t = 0$ would give us the value at β. To learn the restriction to the line, it suffices to learn the restriction on at least $d+1$ points on the line, where d is the degree of the t variable. By evaluating at $d+1$ random points (which can be done since these points look random) on the line, we can accomplish this.

3 Notations and Preliminaries

Throughout the paper, we use X, Y uppercase denote a set of variables, lowercase x_i denotes variables and $\bar{x}, \bar{y}$ to denote vector/tuple of variables and $\bar{v}$ denotes a vector/tuple of field constants. We sometimes abuse notations by referring to a circuit as a collection of multiplication $\Sigma\Pi$ gates. For any circuit $\Sigma\wedge\Sigma(k)$ or $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ or multilinear $\Sigma\Pi\Sigma(k)$, we say that circuit is optimal circuit computing a particular polynomial(say f) if no circuit (in that respective class) can compute f with a smaller fan-in.

3.1 Algebraic Tool Kit

Let $\mathbb{F}$ denote a field, finite or otherwise, and let $\overline{\mathbb{F}}$ denote its algebraic closure.

3.2 Polynomials

A polynomial $f \in \mathbb{F}[x_1, x_2, \ldots, x_n]$ *depends* on a variable x_i if there are two inputs $\bar{\alpha}, \bar{\beta} \in \overline{\mathbb{F}}^n$ differing only in the i^{th} coordinate for which $f(\bar{\alpha}) \neq f(\bar{\beta})$. Equivalently, f *depends* on a variable x_i if there is a monomial in f which contains x_i.

We denote by $\mathrm{var}(f)$ the set of variables that f depends on. We say that f is g are *similar* and denote by it $f \sim g$ if $f = \alpha g$ for some $\alpha \neq 0 \in \mathbb{F}$.

For a polynomial $f(x_1, \ldots, x_n)$, a variable x_i and a field element α, we denote with $f|_{x_i=\alpha}$ the polynomial resulting from substituting α to x_i. Similarly given a subset $I \subseteq [n]$ and an assignment $\bar{a} \in \mathbb{F}^n$, we define $f|_{\bar{x}_I=\bar{a}_I}$ to be the polynomial resulting from substituting a_i to x_i for every $i \in I$.

Let $f, g \in \mathbb{F}[x_1, x_2, \ldots, x_n]$ be polynomials. We say that g *divides* f, or equivalently g is a factor of f, and denote it by $g \mid f$ if there exists a polynomial $h \in \mathbb{F}[x_1, x_2, \ldots, x_n]$ such that $f = g \cdot h$. We say that f is *irreducible* if f is non-constant and cannot be written as a product of two non-constant polynomials.

Given the notion of divisibility, we define the gcd of a set of polynomials in the natural way: we define it to be the highest degree polynomial dividing them all (suitably scaled)[4]. A *linear function* is a polynomial of the form $L(X) = \sum_{i=1}^{n} a_i x_i + a_0$ with $a_i \in \mathbb{F}$. The following folklore lemma expresses a condition for two non-similar linear functions to remain non-similar under a (partial) substitution.

Lemma 3.1 (Folklore). *Let $L(X) = \sum_{i=1}^{n} a_i x_i + a_0$ and $R(X) = \sum_{i=1}^{n} b_i x_i + b_0$ be two linear functions in $\mathbb{F}[x_1, x_2, \ldots, x_n]$ such that $L \nsim R$. Let*

$$D(L,R)(X) \triangleq \prod_{i=1}^{n} \left(a_i R(X) - b_i L(X)\right), \text{ where the product is taken only over non-zero elements.}$$

Let $\bar{u} \in \mathbb{F}^n$ such that $D(L,R)(\bar{u}) \neq 0$. Then for every $I \subsetneq [n]$ it holds that $L|_{\bar{x}_I = \bar{u}_I} \nsim R|_{\bar{x}_I = \bar{u}_I}$.

The interested reader can refer to [SV18] for a proof.

Definition 3.2 (Hybrids & Lines). *Let $\bar{a}, \bar{b} \in \mathbb{F}^n$ and $0 \leq i \leq n$. We define the i-th* hybrid of $\bar{a}, \bar{b}$ *as $\gamma^i(\bar{a}, \bar{b}) \triangleq (b_1, \ldots, b_i, a_{i+1}, \ldots, a_n)$. In particular, $\gamma^0(\bar{a}, \bar{b}) = \bar{a}$ and $\gamma^n(\bar{a}, \bar{b}) = \bar{b}$.*
We define a line *passing through $\bar{a}$ and $\bar{b}$ as $\ell_{\bar{a},\bar{b}} : \mathbb{F} \to \mathbb{F}^n$, $\ell_{\bar{a},\bar{b}}(t) \triangleq (1-t)\cdot\bar{a} + t\cdot\bar{b}$. In particular, $\ell_{\bar{a},\bar{b}}(0) = \bar{a}$ and $\ell_{\bar{a},\bar{b}}(1) = \bar{b}$.*

We state below a well known result by Berlekamp and Welch which gives an efficient algorithm for noisy polynomial interpolation.

Lemma 3.3 (Berlekamp-Welch Algorithm (for a description see [Sud98])). *Let $P(t)$ be a univariate polynomial of degree at most d. There exists a deterministic algorithm that given m evaluations of P with at most e errors outputs $P(t)$, provided that $m - d > 2e + 1$.*

For two vectors $\bar{a}$ and $\bar{b} \in \mathbb{F}^n$, let $\mathrm{w_H}(\bar{a}, \bar{b})$ denote the Hamming distance between $\bar{a}$ and $\bar{b}$

3.3 Partial Derivatives

The concept of a *partial derivative* of a multivariate function and its properties are well-known and well-studied for continuous domains (such as, $\mathbb{R}$, $\mathbb{C}$ etc.). This concept can be extended to polynomials and rational functions over arbitrary fields from a purely algebraic point of view. For more details we refer to reader to [Kap57].

Definition 3.4. *For a monomial $M = \alpha \cdot x_1^{e_1} \cdots x_i^{e_i} \cdots x_n^{e_n} \in \mathbb{F}[x_1, x_2, \ldots, x_n]$ and a variable x_i we define the* partial derivative *of M with respect to x_i, as $\frac{\partial M}{\partial x_i} \triangleq \alpha e_i \cdot x_1^{e_1} \cdots x_i^{e_i - 1} \cdots x_n^{e_n}$. The definition can be extended to $\mathbb{F}[x_1, x_2, \ldots, x_n]$ by imposing linearity and to $\mathbb{F}(x_1, x_2, \ldots, x_n)$ via the quotient rule.*

Observe that the sum, product, quotient and chain rules carry over. In addition, when $\mathbb{F} = \mathbb{R}$ or $\mathbb{F} = \mathbb{C}$ the definition coincides with the analytical one. The following set of rational function plays an important role.

Inspired by a similar notion of [KS09a], we define a distance measure between multiplication gates. This measure will play a crucial role in the analysis of our reconstruction algorithm. Roughly speaking, the distance between two polynomials, each of them being product of linear forms, is the number of factors that appear in only one of them.

Definition 3.5 (Distance). *For $f, g \in \mathbb{F}[x_1, x_2, \ldots, x_n]$, we define a* distance *function:*

$$\Delta(f,g) \triangleq \frac{\max\{\deg(f), \deg(g)\}}{\deg(\gcd(f,g))}.$$

[4]Such a polynomial is unique up to scaling, and one can fix a canonical polynomial in this class for instance by requiring that the leading monomial has coefficient 1. With this definition, two polynomials are pairwise coprime if their gcd is of degree 0, and in particular the gcd equals 1.

3.4 Depth-3 Circuits

In this section we formally introduce the general model of depth-3 circuits and specialization of set-multilinear depth-3 circuits, which is the focus of our paper. It is to be noted that depth-3 circuits were a subject for a long line of study [DS07, KS07, KS09b, SV15, AM10, KS11, SS11, SS12, SS13].

Definition 3.6. *A depth-3 $\Sigma\Pi\Sigma(k)$ circuit C computes a polynomial of the form*

$$C(X) = \sum_{i=1}^{k} T_i(X) = \sum_{i=1}^{k} \prod_{j=1}^{d_i} \ell_{i,j}(X),$$

where the $\ell_{i,j}$-s are linear functions; $\ell_{i,j}(X) = \sum_{t=1}^{n} a_{i,j}^t x_t + a_{i,j}^0$ with $a_{i,j}^t \in \mathbb{F}$.
A multilinear $\Sigma\Pi\Sigma(k)$ *circuit is a $\Sigma\Pi\Sigma(k)$ circuit in which each T_i is a multilinear polynomial. In particular, each such T_i is a product of variable-disjoint linear functions.*
Given a partition $X = \sqcup_{j\in[d]} X_j$ of X, a set-multilinear $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ *circuit is a further specialization of a multilinear circuit to the case when each $\ell_{i,j}$ is a linear* form *in $\mathbb{F}[X_j]$. That is, each $\ell_{i,j}$ is defined over the variables in X_j and $a_{i,j}^0 = 0$.*

We say that C is minimal *if no subset of the multiplication gates sums to zero. We define* $\gcd(C)$ *as the linear product of all the non-constant linear functions that belong to all the T_i-s. I.e.* $\gcd(C) = \gcd(T_1, \ldots, T_k)$.
We say that C is simple *if* $\gcd(C) = 1$. *The simplification of C, denoted by* $\mathrm{sim}(C)$, *is defined as $C/\gcd(C)$. In other words, the circuit resulting upon the removal of all the linear functions that appears in* $\gcd(C)$. *Finally, we say that a $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}$ circuit has* width w, *if $|X_j| \leq w$ for all j.*

Throughout the paper, we will be referring to this quantity as the *width* of a polynomial, width of a circuit, since our model is is $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}$ circuits, it all essentially means the same.

3.4.1 Existing Algorithms

We require the following results. In what follows we focus on multilinear and set-multilinear circuits. We begin with polynomial identity testing algorithms. We will first state the well-known Schwartz-Zippel lemma followed by a deterministic black-box identity testing algorithm for multilinear depth-3 circuits. These algorithms will be used in the testing phase. A black-box PIT is an algorithm that tests if a given circuit computes the zero polynomial by only evaluating the circuit on points, and not inspecting the internal structure of the circuit. Hence all that a black-box PIT can do is evaluate the circuit on a small list of points which is guaranteed to have a property that every non-zero circuit produces at least one non-zero evaluation in the list. Such lists are also called hitting sets, the black-box PITs are also called hitting set generators. All the black-box PIT results discussed below can also be interpreted as existence of explicit hitting sets, these hitting sets will be used in derandomizing our learning algorithms.

Lemma 3.7. *[Sch80, Zip79, DL78] Let $f(x_1, ..., x_n)$ be a nonzero polynomial of degree at most d, and let $S \subseteq \mathbb{F}$. If we choose $\bar{a} = (a_1, \ldots, a_n) \in S^n$ uniformly at random, then $Pr[f(\bar{a}) = 0] \leq d/|S|$.*

Lemma 3.8 ([SS12, SV15])**.** *There is a deterministic algorithm that given a black-box access to a multilinear $\Sigma\Pi\Sigma(k)$ circuit C decides if $C \equiv 0$, in time $n^{\mathcal{O}(k)}$.*

The next result provides a factorization algorithm for multilinear depth-3 circuits. A crucial observation is that factors of a multilinear polynomial must be variable-disjoint. Therefore, each factor of a multilinear polynomial P is obtained by restricting P to an appropriate subset of variables.

Lemma 3.9 ([SV10])**.** *There is a deterministic algorithm that given a black-box access to a multilinear/set-multilinear $\Sigma\Pi\Sigma(k)$ circuit C, outputs black-boxes for the irreducible factors of C, in time $n^{\mathcal{O}(k)}$. In addition, each such irreducible factor is computable by a multilinear/set-multilinear $\Sigma\Pi\Sigma(k)$ circuit.*

As a corollary, we can efficiently simulate a black-box access to $\mathrm{sim}(C)$ given a black-box access to C. The main observation is that a linear function that appears in $\gcd(C)$ constitutes an irreducible factor of C.

Corollary 3.10. *There is a deterministic algorithm that given a black-box access to a multilinear/set-multilinear $\Sigma\Pi\Sigma(k)$ circuit C outputs linear functions $L_1, \ldots, L_r$ and black-box access to a simple multilinear/set-multilinear $\Sigma\Pi\Sigma(k)$ circuit $\hat{C}$ such that $C = \prod_{i=1}^{r} L_i \cdot \hat{C}$, in time $n^{\mathcal{O}(k)}$.*

Proof. We describe the following algorithm:

- Run the algorithm from Lemma 3.9 to obtain black-boxes $C_1, \ldots, C_m$ for the irreducible factors of C.
- For each i, try to learn C_i as a linear function by evaluating it on the standard base vectors and $\bar{0}$. Let L_i denote the resulting purported linear function.
- As each C_i is computable by a multilinear $\Sigma\Pi\Sigma(k)$ circuit, use the identity testing algorithm from Lemma 3.8 on $C_i - L_i$ to determine which C_i-s compute linear functions.
- Wlog, let $C_1, \ldots, C_r$ be the irreducible factors that correspond to linear functions. Set $V \triangleq \bigcup_{i=1}^{r} \mathrm{var}(L_i)$;
- Use Lemma 3.8 to find an assignment $\bar{a} \in \mathbb{F}^X$ such that $C(\bar{a}) \neq 0$.
- Output: $L_1, \ldots, L_r, \hat{C} \triangleq \frac{C|_{X_V = \bar{a}_V}}{\prod_{i=1}^{r} L_i(\bar{a}_V)}$

The claim regarding the runtime follows from Lemmas 3.8 and 3.9. For the analysis, observe that $L_1, \ldots, L_r$ are factors of C. In addition, as C computes a multilinear polynomial, its factors are variable-disjoint. Therefore, we can write

$$C(V, X \setminus V) = \prod_{i=1}^{r} L_i \cdot C'(X \setminus V).$$

Consequently:

$$C(\bar{a}_V, X \setminus V) = \prod_{i=1}^{r} L_i(\bar{a}_V) \cdot C'(X \setminus V).$$

and

$$C(X \setminus V) = \prod_{i=1}^{r} L_i \cdot \frac{C(\bar{a}_V, X \setminus V)}{\prod_{i=1}^{r} L_i(\bar{a}_V)} = \prod_{i=1}^{r} L_i \cdot \hat{C}.$$

Note that $\prod_{i=1}^{r} L_i(\bar{a}_V) \neq 0$ as $C(\bar{a}) \neq 0$. Finally, since every factors of $\hat{C}$ constitutes a factor of C, and all the linear factors of C has been accounted for it follows that $\hat{C}$ has no linear factors.

□

3.4.2 Structural Results

In this we discuss a strong structural result about set-multilinear depth-3 circuits computing the zero polynomial. We note that results of this flavor were proven before for more general families of depth-3 circuits (for more details see e.g. [DS07, KS09b, SS13] and references within). We prove our result by a reduction to the case where each linear function is, in fact, a univariate polynomial.

Lemma 3.11 ([AvMV15])**.** *Let $k \geq 2$ and let $C \equiv \sum_{i=1}^{k} T_i = \sum_{i=1}^{k} \prod_{j=1}^{d_i} \ell_{i,j}$ be a simple and minimal multilinear circuit $\Sigma\Pi\Sigma(k)$ circuit where each $\ell_{i,j}$ is a univariate polynomial. If C computes the zero polynomial then for all $i \in [k] : |\mathrm{var}(T_i)| \leq k - 2$.*

Theorem 3.12. *Let $C \equiv \sum_{i=1}^{k} T_i$ be a simple and minimal $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}$ circuit computing the zero polynomial. Then for all $i \in [k] : \deg(T_i) \leq k-2$.*

Proof. Fix $i \in [k]$. Recall that T_i is of the form $T_i = \prod_{j=1}^{d} \ell_{i,j}(X_j)$. Fix $j \in [d]$. Pick a variable $x_j \in \text{var}(\ell_{i,j})$ and let $\bar{a}_j \in \mathbb{F}^{X_j \setminus \{x_j\}}$ be a random assignment to the variables $X_j \setminus \{x_j\}$. As C is simple, $\gcd(\ell_{1,j}, \ldots, \ell_{k,j}) = 1$. By the choice of $\bar{a}_j$ we obtain that $\gcd\left(\ell_{1,j}(\bar{a}_j, x_j), \ldots, \ell_{k,j}(\bar{a}_j, x_j)\right) = 1$. Now consider the circuit $\hat{C} \equiv \sum_{i=1}^{k} \hat{T}_i$ obtained from C be assigning each set $X_j \setminus \{x_j\}$ to $\bar{a}_j$. Observe that $\hat{C}$ satisfies the premises of Lemma 3.11. Moreover, as $\hat{T}_i$ is a product of univariate polynomials, $\deg(T_i) = \deg(\hat{T}_i) = \left|\text{var}(\hat{T}_i)\right|$. Therefore, $\deg(T_i) = \left|\text{var}(\hat{T}_i)\right| \leq k-2$, as required. □

This structural result, in turn, implies that the distance (see Definition 3.5) between two multiplication gates in a minimal circuit, computing the zero polynomial, is "small".

Lemma 3.13. *Let $C \equiv \sum_{i=1}^{k} T_i$ be a minimal $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}$ circuit computing the zero polynomial. Then for all $i \neq j \in [k] : \Delta(T_i, T_j) \leq k-2$.*

Proof. Consider $\text{sim}(C)$. By definition, $\text{sim}(C)$ is simple. In addition, observe that $\text{sim}(C)$ minimal and computes the zero polynomial, by construction. By Theorem 3.12, for each $i \in [k]$ we have

$$\frac{\deg(T_i)}{\deg(\gcd(C))} = \deg\left(\frac{T_i}{\gcd(C)}\right) \leq k-2.$$

Therefore,

$$\Delta(T_i, T_j) = \frac{\max\{\deg(T_i), \deg(T_j)\}}{\deg(\gcd(T_i, T_j))} \leq \frac{(k-2) \cdot \deg(\gcd(C))}{\deg(\gcd(T_i, T_j))} \leq k-2.$$

The last inequality follows from the fact that $\gcd(T_i, T_j)$ divides $\gcd(C)$. □

The above result implies that any pair of circuits computing the same polynomials must have "many" common linear functions. These results are also refereed as rank-bounds in the literature. Also, for our applications we don't need $\Delta(T_i, T_j) \leq k-2$ at this granular detail, we will simply upper bound this by k.

Lemma 3.14. *$C \equiv T_1 + T_2 \ldots T_k$ and $C' \equiv T'_1 + T'_2 \ldots T'_{k'}$ be two $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}$ circuits computing the same polynomial with $k' \leq k$. Furthermore, suppose C is minimal. Then for each T_i (in C) there exists a T'_j (in C') such that $\Delta(T_i, T'_j) \leq k$.*

Proof. Notice that, $C - C' \equiv 0$, that is $T_1 + T_2 + \ldots + T_k - T'_1 - T'_2 - \ldots - T_{k'} = 0$. Pick $i \in [k]$ and let C_i be a minimal subcircuit computing the zero polynomial that contains T_i. As C is a minimal circuit, C_i must contain at least one of T'_j-s. The result now follows directly from Theorem 3.12. □

Other useful lemmas

Definition 3.15. *Let $\boldsymbol{f} := (f_1, f_2, \ldots f_m)$, where $f_i(X) \in \mathbb{F}[\boldsymbol{X}]$, be a vector of polynomials over a field $\mathbb{F}$. The set of $\mathbb{F}$-linear dependencies in $\boldsymbol{f}$, denoted $\boldsymbol{f}^{\perp}$, is the set of all vectors $\boldsymbol{v} \in \mathbb{F}^m$ whose inner product with $\boldsymbol{f}$ is the zero polynomial, i.e.,*

$$\boldsymbol{f}^{\perp}\{(a_1, \ldots, a_m) \in \mathbb{F}^m : a_1 f_1(\boldsymbol{X}) + \ldots + a_m f_m(\boldsymbol{X}) = 0\}.$$

The set $\mathbf{f}^{\perp}$ is clearly a linear subspace of $\mathbb{F}^m$. This notion is helpful to state and prove some useful lemmas. The main observation here is that given (by arithmetic circuits or black-box access) a collection of polynomials then in randomized polynomial time we can find the $\mathbb{F}$-linear dependencies among these polynomials. In order to show that we will need the following technical lemma.

Lemma 3.16. *[Kay11, Lem 4.1] Let $f_1, f_2, \dots f_k \in \mathbb{F}[x_1, \dots, x_n]$ be $\mathbb{F}$-linearly independent polynomials with $|\mathbb{F}| > nkd$ (otherwise we can work with an extension) and $a_1, \dots, a_k \in \mathbb{F}^n$ be k random points[5] then the following matrix has full rank with high probability.*

$$M = \begin{pmatrix} f_1(a_1) & f_2(a_1) & \cdots & f_k(a_1) \\ f_1(a_2) & f_2(a_2) & \cdots & f_k(a_2) \\ \vdots & \vdots & \vdots & \vdots \\ f_1(a_k) & f_2(a_k) & \cdots & f_k(a_k) \end{pmatrix}$$

Proof. Let $X_1, \dots, X_k$ be disjoint sets of variables each of size n. Define,

$$Q = \begin{pmatrix} f_1(X_1) & f_2(X_1) & \cdots & f_k(X_1) \\ f_1(X_2) & f_2(X_2) & \cdots & f_k(X_2) \\ \vdots & \vdots & \vdots & \vdots \\ f_1(X_k) & f_2(X_k) & \cdots & f_k(X_k) \end{pmatrix}. \tag{2}$$

We will show, via induction on k, that Q has full rank, or equivalently, the determinant of Q is a nonzero polynomial. The result will then follow via the Schwartz-Zippel Lemma applied to the determinant of Q.

Note that $k = 1$ follows directly. On expanding $Det(Q)$ along the first row we get,

$$Det(Q) = \sum_{j=1}^{k} (-1)^{j+1} f_j(X_1) Q_{1j}$$

where Q_{ij} is the determinant of the ij-th minor. Notice that every Q_{1j} , $j \in [k]$, is a polynomial in the set of variables $X_2, ..., X_k$. By induction, every Q_{1j} is a nonzero polynomial (since every subset of a set of $\mathbb{F}$-linearly independent polynomials is also $\mathbb{F}$-linearly independent). If $Det(Q)$ was the zero polynomial then plugging in random values for $X_2, ..., X_k$ would give us a nonzero $\mathbb{F}$-linear dependence among $f_1(X_1), f_2(X_1), ..., f_k(X_1)$, which is a contradiction. Hence $Det(Q)$ must be nonzero, proving the claim. This along with Lemma 3.7 gives that M is invertible with high probability. □

Once we have the above lemma, one can easily use it to determine the linear dependency structure of a set of polynomials as in the next lemma.

Lemma 3.17. *[Kay11, Lem 4.1] Given m polynomials $\boldsymbol{f} = \{f_1, f_2, \dots f_m\}$, each in $\mathbb{F}[x_1, \dots, x_n]$ of degree at most d, either by a circuit(or black-box access) [6], s.t. with rank(maximal number of linearly independent f_i-s) of $\boldsymbol{f} = k$, and $|\mathbb{F}| > \binom{m}{k} \cdot dnk$ (if $|\mathbb{F}| \leq \binom{m}{k} \cdot dnk$ then we can work with an extension) then:*

1. *There is a randomized* $\mathrm{poly}(m, n, k)$ *time algorithm to compute a basis for the space $\mathbb{F}$-span$\{f_1, f_2, \dots f_m\}$. Along with the basis(say $f_{i_1}, f_{i_2}, \dots f_{i_k}$ be a basis of linear space of f_i-s), the aforementioned algorithm also outputs a matrix M s.t.*

$$M \begin{pmatrix} f_{i_1} \\ \vdots \\ f_{i_k} \end{pmatrix} = \begin{pmatrix} f_1 \\ \vdots \\ f_m \end{pmatrix}.$$

2. *Also, there is a randomized* $\mathrm{poly}(m, n, k)$ *time algorithm that given a vector of these polynomials $\boldsymbol{f} = (f_1, f_2, \dots, f_m)$ computes a basis for the space $\boldsymbol{f}^{\perp}$.*

[5] More precisely, let $S \subseteq \mathbb{F}$ be a set pf size nkd and let each a_i be chosen independently and uniformly at random from S^n

[6] The lemma statement in [Kay11] just mentions the case when a circuit is given explicitly, however it is easy to observe that even black-box/oracle access suffices.

Proof Sketch. Let $X_1, \ldots, X_k$ be disjoint sets of variables each of size n. Define,

$$Q = \begin{pmatrix} f_1(X_1) & f_2(X_1) & \cdots & \cdots & f_m(X_1) \\ f_1(X_2) & f_2(X_2) & \cdots & \cdots & f_m(X_2) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ f_1(X_k) & f_2(X_k) & \cdots & \cdots & f_m(X_k) \end{pmatrix}.$$

The crucial observation here is that linear dependencies of the f_i are exactly captured by linear dependencies of the columns of Q and moreover this continues to hold after substituting random values to the X_i-s. We will show in the next paragraph how to prove this. Note that once we have this fact then we have reduced the problem of determining the linear dependencies that hold between the polynomials to determining the linear dependencies that hold between vectors in $\mathbb{F}^m$, and for vectors in $\mathbb{F}^m$ we do know efficient algorithms for computing the basis, the orthogonal subspace and the suitable matrix M.

In order to see that linear dependencies between the f_i are captured by linear dependencies among the columns of Q after the random substitutions, it suffices to show that for any size k subset of f_i-s which is linearly independent, the corresponding minor of Q has full rank. Note that there can be at most $\binom{m}{k}$ such full rank minors, and we have to ensure that the determinant of each full rank minor stays nonzero, which in-turn boils down to "hitting" (finding a non-zero assignment) the product of these determinants. Note that the degree of the product of such determinants is bounded by $\binom{m}{k} \cdot dnk$. Thus, by Lemma 3.7 we get that random substitutions (given $|\mathbb{F}| > \binom{m}{k} \cdot dnk$) ensure that with high probability, for any subset of f_i-s which are linearly independent, the corresponding minor of Q has full rank. □

Interestingly, when f_i-s are from special classes of polynomials for which deterministic blackbox PIT algorithms (explicit hitting sets) are known, then we can derandomize the previous lemma. Concretely, if $f_1, \ldots, f_m$ have rank $k = \mathcal{O}(1)$ with each $f_i \in \mathcal{C}$, where $\mathcal{C}$ is an arithmetic circuit class. Then the randomized algorithms in Lemma 3.17 and Lemma 3.16 can be derandomized given polynomial sized hitting sets for the class $\underbrace{\mathcal{C} + \ldots + \mathcal{C}}_{k \text{ times}}$. Here, $\underbrace{\mathcal{C} + \ldots + \mathcal{C}}_{k \text{ times}}$ is a circuit class which comprises of $\mathbb{F}$-linear combinations of k polynomials in $\mathcal{C}$. Also, for our applications $\mathcal{C}$ will either be $\Sigma \wedge \Sigma(k')$ circuits, $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k')$ circuits or multilinear $\Sigma\Pi\Sigma(k')$ circuits with $k = \mathcal{O}(1)$ and $k' = \mathcal{O}(1)$, and thus we do have such hitting sets.

We will start by the stating deterministic version of Lemma 3.16.

Lemma 3.18. *Let $f_1, f_2, \ldots f_k \in \mathbb{F}[x_1, \ldots, x_n]$ be $\mathbb{F}$-linearly independent polynomials with $|\mathbb{F}| > nkd$ (otherwise we can work with an extension) and $f_i \in \mathcal{C}$, where $\mathcal{C}$ is an arithmetic circuit class. Furthermore, let $\mathcal{H}_k$ be a hitting set for the class $\underbrace{\mathcal{C} + \ldots + \mathcal{C}}_{k \text{ times}}$. Then there exist $(a_1, a_2, \ldots, a_k)$ with each $a_i \in \mathcal{H}_k$ s.t. the following matrix has full rank.*

$$M = \begin{pmatrix} f_1(a_1) & f_2(a_1) & \cdots & f_k(a_1) \\ f_1(a_2) & f_2(a_2) & \cdots & f_k(a_2) \\ \vdots & \vdots & \vdots & \vdots \\ f_1(a_k) & f_2(a_k) & \cdots & f_k(a_k) \end{pmatrix}$$

Equivalently, $\underbrace{\mathcal{H}_k \oplus \mathcal{H}_k \cdots \oplus \mathcal{H}_k}_{k \text{ times}}$ is a hitting set for $Det(Q)$, where Q is defined by Eq. 2.

Proof. This follows by essentially the same inductive argument as in the proof of Lemma 3.16. The case $k = 1$ follows directly. For $k > 1$, on expanding $Det(Q)$ along the first row we get, $Det(Q) = \sum_{j=1}^{k} (-1)^{j+1} f_j(X_1) Q_{1j}$, where Q_{ij} is the determinant of the ij-th minor. By induction, every Q_{1j} is a nonzero polynomial and we can deterministically choose $a_2, \ldots, a_n$ using hitting sets for the class $\underbrace{\mathcal{C} + \ldots + \mathcal{C}}_{k-1 \text{ times}}$. For brevity we will refer to the substitution $X_2 = a_2, \ldots, X_n = a_n$ by σ. Thus to find a_1 s.t. M is invertible,

we have to do identity testing for $Det(Q)|_\sigma = \sum_{j=1}^{k}(-1)^{j+1} f_j(X_1)Q_{1j}|_\sigma$ which lies in class $\underbrace{\mathcal{C}+\ldots+\mathcal{C}}_{k \text{ times}}$. This can be done by choosing a_1 from a hitting set for the class $\underbrace{\mathcal{C}+\ldots+\mathcal{C}}_{k \text{ times}}$. □

Using the above derandomized lemma, we show how to derandomize Lemma 3.17 in polynomial time when the number of independent f_i-s is constant.

Lemma 3.19. *Given m polynomials $\boldsymbol{f} = \{f_1, f_2, \ldots f_m\}$, each in $\mathbb{F}[x_1, \ldots, x_n]$ of degree at most d, either by a circuit(or black-box access) s.t. rank(maximal number of linearly independent tuples) of $\boldsymbol{f} = k$, and $f_i \in \mathcal{C}$, where $\mathcal{C}$ is an arithmetic circuit class. Also, let $\mathcal{H}_k$ be a hitting set for the class $\underbrace{\mathcal{C}+\ldots+\mathcal{C}}_{k \text{ times}}$. Then,*

1. *There is a deterministic $\text{poly}(|\mathcal{H}_k|\binom{m}{k}, n, d, m)$ time algorithm to compute a basis for the space $\mathbb{F}$-span$\{f_1, f_2, \ldots f_m\}$. Along with the basis(say $f_{i_1}, f_{i_2}, \ldots f_{i_k}$ be a basis of linear space of f_i-s), the aforementioned algorithm also outputs a matrix M s.t.*

$$M \begin{pmatrix} f_{i_1} \\ \vdots \\ f_{i_c} \end{pmatrix} = \begin{pmatrix} f_1 \\ \vdots \\ f_m \end{pmatrix}.$$

2. *Also, there is a deterministic $\text{poly}(|\mathcal{H}_k| \cdot \binom{m}{k}, n, d, m)$ time algorithm that given a vector of these polynomials $\boldsymbol{f} = (f_1, f_2, \ldots, f_m)$ computes a basis for the space $\boldsymbol{f}^\perp$.*

Proof Sketch. Let $X_1, \ldots, X_k$ be disjoint sets of variables each of size n. Define,

$$Q = \begin{pmatrix} f_1(X_1) & f_2(X_1) & \cdots & \cdots & f_m(X_1) \\ f_1(X_2) & f_2(X_2) & \cdots & \cdots & f_m(X_2) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ f_1(X_k) & f_2(X_k) & \cdots & \cdots & f_m(X_k) \end{pmatrix}.$$

The goal is the derandomize the algorithm from Lemma 3.17. In order to do this, we have to ensure that for any size k subset of f_i-s which are linearly independent, there is a deterministic substitution of the variables the corresponding minor has full rank. Note that there can be at most $\binom{m}{k}$ full rank minors and we have to ensure that we find a deterministic substitution of the variables such that the determinant of each full rank minor stays nonzero, which is equivalent to keeping the product of such determinants nonzero after substitution. By Lemma 3.18, there is a polynomial sized hittting sit for the determinant for each of these full rank minors. Along with standard blackbox PIT trick of working with hitting set "generators", we can find a hitting set of size $\text{poly}(|\mathcal{H}_k| \cdot \binom{m}{k})$ for the product of the determinants of each full rank minor (see [SY10, Sec. 4.1] for details).

Once we have the hitting set, we can then choose that element of the hitting set that maximizes the number of $k \times k$ minors whose determinant is nonzero after substitution to find the appropriate substitution such that the columns of Q will have the same linearly dependency structure as the given polynomials. □

Lemma 3.20. *Set-multilinear $\Sigma\Pi\Sigma(k)$ circuits are closed under factoring. That is, if $f = g \cdot h$ and f is computed by a $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuit. Then g (similarly h) is computed by a set-multilinear $\Sigma\Pi\Sigma(k)$ circuit. Also, there is a partition of $[d]$ into two disjoint sets A_1 and A_2 s.t. g is set-multilinear w.r.t to partition $\sqcup_{i\in A_1} X_i$ and h is set-multilinear w.r.t. to partition $\sqcup_{i\in A_2} X_i$.*

Proof. Let $f = g \cdot h$ where has f has a $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuit, say C_f.

We will first show that g, h are set-multilinear with two variable disjoint partitions. That is variables from a partition either occur in g or occur in h. Formally, for each X_i, if a variable $v \in \text{var}(g) \cap X_i$, then $X_i \cup \text{var}(h) = \phi$. Suppose, for contradiction, $\exists X_i$ s.t $u, v \in X_i$ s.t. $u \in \text{var}(g)$ and $v \in \text{var}(h)$. That is,

on writing g, h as univariates in u, v respectively, we get $g = a_d u^d + \ldots + a_o$ and $h = b_{d'} v^{d'} + \ldots + b_o$ with $d, d' \geq 1$ and $a_d, b_{d'} \neq 0$. Now, notice that coefficient of $u^d v^{d'} \neq 0$ in $f = g \cdot h$ thus contracting the assumption that f was set-multilinear to begin with.

The proof concludes by setting all variables in h to random values(s.t. h doesn't evaluate to 0) in C_f and observing that the resulting circuit is a set-multilinear circuit computing a constant multiple of g. □

Lemma 3.21. *Let $C_f \equiv T_1 + T_2 + \cdots + T_k$ be an optimal degree d $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuit computing f, where $T_i = \prod_{j \in [d]} \ell_{i,j}(X_j)$. Then $\forall\, j \in [d]$ the polynomials in the set $S_j := \{\frac{T_1}{\ell_{1,j}}, \frac{T_2}{\ell_{2,j}}, \ldots, \frac{T_k}{\ell_{k,j}}\}$ are linearly independent. Note that, S_j is a set of polynomials (not rational functions) because $\ell_{i,j} | T_i \, \forall j \in [k], i \in [d]$.*

Proof. Assume on contrary that $\exists j$ s.t. the linear forms in $S_j = \{T_1/\ell_{1,j}, T_2/\ell_{2,j}, \ldots T_k/\ell_{k,i}\}$ has a linear dependence. Let $\sum_{i=1}^{k-1} \lambda_i T_i/\ell_{i,j} = T_k/\ell_{k,j}$, be a non-trivial linear dependence. Note that, this can be assumed always by just relabelling the gates. This implies that,

$$f(\bar{x}) = \sum_{i=1}^{k-1} (\ell_{i,j} + \lambda_i \ell_{k,j}) \cdot T_i/\ell_{i,j}. \tag{3}$$

Note that, equation 3 is a $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k-1)$ representation of f, thus contradicting our assuption that C_f is optimal. □

3.5 Variable Reduction

In this section we discuss how to reduce the number of variables in a polynomial. Before describing this procedure we have to formally define the notion of number of essential variables in a polynomial.

Definition 3.22 (number of essential variables)**.** *For $f(\bar{x}) \in \mathbb{F}[\bar{x}]$, we will say that the number of essential variables in $f(\bar{x})$ is t if there exist an invertible linear transformation $A \in \mathbb{F}^{n \times n}$ s.t. $f(A\bar{x})$ just depends on t variables.*

The next lemma is from the work of Carlini [Car06], adopted by Kayal [Kay11] in the language of circuits. This lemma eliminates redundant variables from a polynomial and plays a crucial role in our reconstruction results for $\Sigma\wedge\Sigma$ and multilinear-$\Sigma\Pi\Sigma$ circuits.

We state the lemma below in the setting of black-box access to the input polynomial. The original version of the lemma was in the whitebox setting, but by inspecting the proof in [Kay11] one can see that it works in the black-box setting as well by noting that given black-box access to a circuit computing a polynomial f, one can get black-box access to the circuits computing its first order partial derivatives.

Lemma 3.23. *[Kay11, Car06] Given black-box access to an n-variate polynomial $f(X) \in \mathbb{F}[X]$ of degree d with m essential variables, s.t. $char(\mathbb{F}) > d$ or 0, there is a randomized $\mathrm{poly}(n, d, s)$ time algorithm (where s is the size of the circuit computing f) that computes an invertible linear transformation $A \in \mathbb{F}^{(n \times n)}$ such that $f(A \cdot \bar{x})$ depends on the first m-variables only.*

Interestingly, when f is from a special class of polynomials for which explicit hitting sets are known even for first order partial derivatives of f, and additionally if the number of essential variables in f is small, then we can derandomize the previous lemma. It is worth nothing that many or most interesting classes of circuits are closed under taking partial derivatives.

Lemma 3.24. *Let $\mathcal{C}$ be class of arithmetic circuits that is closed under first order partial derivatives. We are given black-box access to an n-variate polynomial $f(X) \in \mathbb{F}[x_1, \ldots, x_n]$ of degree d, computable by a size s circuit in $\mathcal{C}$, such that f has k essential variables and $char(\mathbb{F}) > d$ or 0. Let $\mathcal{H}_k$ be a hitting set for the class $\underbrace{\mathcal{C} + \ldots + \mathcal{C}}_{k \text{ times}}$. Then there is a deterministic $\mathrm{poly}(\binom{n}{k}, d, s, |\mathcal{H}_k|)$ time algorithm that computes an invertible linear transformation $A \in \mathbb{F}^{(n \times n)}$ such that $f(A \cdot \bar{x})$ depends on the first k-variables only.*

Proof Sketch. The proof of this lemma is obtained by derandomizing the proof of Lemma 3.23 in the current setting. Since we haven't provided the details of the proof of Lemma 3.23, we will only provide a proof sketch here of the changes needed to be made to the proof of Lemma 3.23 to derandomize it. The only place where randomness is used in the proof of Lemma 3.23 is in computing a basis of $\mathbf{f}^{\perp}$, where $\mathbf{f} := \left(\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n}\right)$. As in the proof of Lemma 3.23, one observes that, rank of $\mathbf{f}$ = number of essential variables $= k$. Once we have this, then the assumption that $\mathcal{C}$ is closed under taking first order partial derivatives, along with the hitting set $\mathcal{H}_k$ satisfies all the preconditions for Lemma 3.19. Thus, we can use Lemma 3.19 to compute a basis for $\mathbf{f}^{\perp}$ deterministically, which in turn gives a deterministic algorithm for computing A s.t. $f(A \cdot \bar{x})$ depends on the first k-variables only. The time complexity follows directly. □

3.6 Tensors and Set-Multilinear Depth-3 Circuits

Tensors, higher dimensional analogues of matrices, are multi-dimensional arrays with entries from some field $\mathbb{F}$. For instance, a 3-dimensional tensor can be written as $\mathcal{T} = (\alpha_{i,j,k}) \in \mathbb{F}^{n_1 \times n_2 \times n_3}$ and 2-dimensional tensors simply corresponds to traditional matrices. We will work with general d-dimensional tensors $\mathcal{T} = (\alpha_{j_1,j_2,\ldots,j_d}) \in \mathbb{F}^{n_1 \times \cdots \times n_d}$, here $[n_1] \times \cdots \times [n_d]$ refers to the shape of the tensor and n_i as length of tensor in i-th dimension. Just like any matrix has a natural definition of rank, there is an analogue for tensors as well.

The rank of a tensor $\mathcal{T}$ can be defined as the smallest r for which $\mathcal{T}$ can be written as a sum of r tensors of rank 1, where a rank-1 tensor is a tensor of the form $v_1 \otimes \cdots \otimes v_d$ with $v_i \in \mathbb{F}^{n_i}$. Here $\otimes$ is the Kronecker (outer) product a.k.a *tensor product*. The expression of $\mathcal{T}$ as a sum of such rank-1 tensors, over the field $\mathbb{F}$ is called $\mathbb{F}$-*tensor decomposition* or just tensor decomposition, for short. The notion of Tensor rank/decomposition has become a fundamental tool in different branches of modern science with applications in statistics, signal processing, complexity of computation, psychometrics, linguistics and chemometrics. We refer the reader to a monograph by Landsberg [Lan12] and the references therein for more details on application of tensor decomposition.

For our application, it would be useful to think of tensors as a restricted form of multilinear polynomials that are called *set-multilinear* polynomials. To this end, let us fix the following notation throughout the paper.
Let $d \in \mathbb{N}$. We will refer to d as the *dimension*. For $j \in [d]$ let $X_j = \left\{x_{j,1}, x_{j,2}, \ldots, x_{j,n_j}\right\}$, where $n_j = |X_j|$. Finally, let $X = \sqcup_{j \in [d]} X_j$. That is, $\{X_j\}_{\{j \in [d]\}}$ form a partition of X.

Definition 3.25 (Set-Multilinear polynomial)**.** *A polynomial $P \in \mathbb{F}[X]$ is called* set-multilinear *w.r.t (the partition) X, if every monomial that appears in P is of the form $x_{i_1} x_{i_2} \cdots x_{i_d}$ where $x_{i_j} \in X_j$.*

In other words, each monomial of a set-multilinear polynomial picks up exactly one variable from each part in the partition. These polynomial have been well studied in the past [Raz13, FSS14, AKV20] in particular since many natural polynomials like the Determinant, the Permanent, Nisan-Wigderson and others are set-multilinear w.r.t appropriate partitions of variables. Furthermore, each tensor can be regraded as a set-multilinear polynomial.

Definition 3.26. *For a tensor $\mathcal{T} = (\alpha_{j_1,j_2,\ldots,j_d}) \in \mathbb{F}^{n_1 \times \cdots \times n_d}$ consider the following polynomial*

$$f_{\mathcal{T}}(X) \triangleq \sum_{(j_1,\ldots,j_d) \in [n_1] \times \cdots \times [n_d]} \alpha_{j_1,j_2,\ldots,j_d} x_{1,j_1} x_{2,j_2} \cdots x_{d,j_d}.$$

Observe that $f_{\mathcal{T}}(X)$ is a set-multilinear polynomial w.r.t X. More interestingly, there is a direct correspondence between tensor decomposition and computing the polynomial $f_{\mathcal{T}}(X)$ in the model of set-multilinear depth-3 circuits. We first define the model formally.

Definition 3.27 (Set-Multilinear Depth-3 Circuits)**.** *A set-multilinear depth-3 circuit w.r.t to (a partition)*

X with top fan-in k, denoted by $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ computes a (set-multilinear) polynomial of the form

$$C(X) \equiv \sum_{i=1}^{k} \prod_{j=1}^{d} \ell_{i,j}(X_j)$$

where $\ell_{i,j}(X_j)$ is a linear form in $\mathbb{F}[X_j]$.

To gain some intuition, suppose that $f_{\mathcal{T}}(X) = \ell_{i,1}(X_1) \cdot \ell_{i,2}(X_2) \cdots \ell_{i,d}(X_d)$ for some tensor $\mathcal{T}$. We can observe that in this case $\mathcal{T}$ is a rank-1 tensor. Extending this observation, the following provides a formal connection between tensor decomposition and computing the polynomial $f_{\mathcal{T}}(X)$ by set-multilinear depth-3 circuits.

Observation 3.28. *Let $C(X) = \sum_{i=1}^{k} \prod_{j=1}^{d} \ell_{i,j}$ be a set-multilinear depth-3 circuit over $\mathbb{F}$ computing $f_{\mathcal{T}}(X)$ for a tensor $\mathcal{T} = (\alpha_{j_1,j_2,\ldots,j_d}) \in \mathbb{F}^{n_1 \times \cdots \times n_d}$. Then*

$$\mathcal{T} = \sum_{i=1}^{k} \bar{v}(\ell_{i,1}) \otimes \cdots \otimes \bar{v}(\ell_{i,d})$$

where $\bar{v}(\ell_{i,j})$ corresponds to the linear form $\ell_{i,j}$ as an n_j-dimensional vector over $\mathbb{F}$.

Note that this connection is, in fact, a correspondence: any $\mathbb{F}$-tensor decomposition of $\mathcal{T}$ gives a circuit over $\mathbb{F}$. This leads to the following important lemma:

Lemma 3.29. *A tensor $\mathcal{T} = (\alpha_{j_1,j_2,\ldots,j_d}) \in \mathbb{F}^{n_1 \times \cdots \times n_d}$ has rank at most r if and only if $f_{\mathcal{T}}(X)$ can be computed by a $\Sigma\Pi\Sigma_X(r)$ circuit. Therefore, rank of $\mathcal{T}$ is the smallest k for which $f_{\mathcal{T}}(X)$ can be computed by a $\Sigma\Pi\Sigma_X(k)$ circuit.*

Proof. The proof is straightforward. Note that, $\ell_{i,1}(X_1) \cdot \ell_{i,2}(X_2) \cdots \ell_{i,d}(X_d)$ exactly corresponds to a rank-1 tensors. Thus, C_f gives a rank k $\mathbb{F}$-tensor decomposition of $\mathcal{T}$ and any $\mathbb{F}$-tensor decomposition gives a circuit over $\mathbb{F}$. □

3.7 Symmetric Tensors and Sum of Power of Linear Forms

A tensor $\mathcal{T}$ is called symmetric if $X = X_1 = X_2 = \cdots = X_d$ and we have $\mathcal{T}(i_1, i_2, \ldots, i_d) = \mathcal{T}(j_1, j_2, \ldots, j_d)$ whenever $(i_1, i_2, \ldots, i_d)$ is a permutation of $(j_1, j_2, \ldots, j_d)$. Thus, a symmetric tensor is a higher order generalization of a symmetric matrix. Analogous to tensor rank, *symmetric rank* is obtained when the constituting rank-1 tensors are imposed to be themselves symmetric, that is $\bar{v} \otimes \bar{v} \cdots \otimes \bar{v}$.

Definition 3.30. *For a symmetric tensor $\mathcal{T} = (\alpha_{j_1,j_2,\ldots,j_d}) \in \mathbb{F}^{n \times \cdots \times n}$ consider the following polynomial*

$$f_{Sym,\mathcal{T}}(X) \triangleq \sum_{(j_1,\ldots,j_d) \in [n] \times \cdots \times [n]} \alpha_{j_1,j_2,\ldots,j_d} x_{j_1} x_{j_2} \cdots x_{j_d}.$$

Just like in case of general tensors, computing the symmetric rank reduces to finding the optimal top fan-in of a special class of arithmetic circuits, which is sum of power of linear forms ($\Sigma\wedge\Sigma$) circuits defined below.

Definition 3.31 (Sum of power of linear forms)**.** *The Sum of power of linear forms with top fan-in k computes a polynomial of the form $f = \ell_1^d + \cdots \ell_k^d$ where each ℓ_i is a linear polynomial over the n variables.*

Observation 3.32. *Let $C(X) = \sum_{i=1}^{k} \ell_i^d$ be a $\Sigma\wedge\Sigma(k)$ circuit over $\mathbb{F}$ computing $f_{Sym,\mathcal{T}}(X)$ for a symmetric tensor $\mathcal{T} = (\alpha_{j_1,j_2,\ldots,j_d}) \in \mathbb{F}^{n_1 \times \cdots \times n_d}$. Then*

$$\mathcal{T} = \sum_{i=1}^{k} \bar{v}(\ell_i) \otimes \cdots \otimes \bar{v}(\ell_i)$$

where $v(\ell_i)$ is a n-dimensional vector corresponding to the linear form $\ell_{i,j}$.

Remark 3.33. *Both Tensor rank and Symmetric rank are dependent on the underlying field, that is Tensor rank of a tensor $\mathcal{T}$ over $\mathbb{F}$ and $\mathbb{G}$, an extension of $\mathbb{F}$ can be different, see [Shi16, SS16] for details. The correspondence discussed above, among Tensor rank(symmetric rank) and top fan-in of $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}$ circuits($\Sigma\wedge\Sigma$ circuits), respects the dependence of rank on underlying field. That is, in order to find rank of $\mathcal{T}$ over $\mathbb{G}$ we have to find an optimal top fan-in of a $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}$ circuit over $\mathbb{G}$ computing $f_{\mathcal{T}}$.*

3.8 Complexity of Solving a System of Polynomial Equations

Solving a system of polynomial equations is the following problem: For a field $\mathbb{F}$, we are given m polynomials $f_1, f_2, \ldots, f_m \in \mathbb{F}[x_1, \ldots, x_n]$, each of degree at most d. We want to test if there exist a solution (this is the decision version) to $f_1 = 0, f_2 = 0, \ldots, f_m = 0$ in $\mathbb{F}^n$, or find a solution if it exists (this is the search version). A straightforward reduction from 3-SAT shows that polynomial system solving is NP-hard in general. This is a fundamental problem in computational algebra, and it has received lot of attention over various fields. To mention a few, system solving is NP-complete for finite fields, in PSPACE over $\mathbb{R}$ [Can88] and in Polynomial Hierarchy (Σ_2), assuming GRH [Koi96].

Interestingly, for $\mathbb{F} = \mathbb{Q}$ system solving is not even known to be decidable! In fact, if we restrict the question to integral domains (like $\mathbb{Z}$) then the problem is undecidable. This was the well-known Hilbert's tenth problem, which asks if a given Diophantine equation has an integral solution, and was famously proved to be undecidable in the 70's, see [MR75].

In this work, we are mainly concerned with polynomial system solving when the number of variables involved is small (such as a constant). In this case, polynomial system solving turns out is efficient under various settings. We will use the following definitions for describing the complexity of solving a system of equations under various settings.

Definition 3.34 ($\mathrm{Sys}_{\mathbb{F}}(n, m, d)$)**.** *Let $\mathrm{Sys}_{\mathbb{F}}(n, m, d)$ denote the randomized time complexity of finding a solution $\in \mathbb{F}^n$ to a system of m polynomial equations $\in \mathbb{F}[x_1, \ldots, x_n]$ of total degree d (if one exists).*

Also, consider a weaker version of the above problem, let $\widetilde{\mathrm{Sys}}_{\mathbb{F}}(n, m, d)$ denote the randomized time complexity of finding a solution (could be in an extension of $\mathbb{F}$) to a system of m polynomial equations $\in \mathbb{F}[x_1, \ldots, x_n]$ of total degree d (if one exists).

Definition 3.35 ($\text{Det-}\mathrm{Sys}_{\mathbb{F}}(n, m, d)$)**.** *Let Det-$\mathrm{Sys}_{\mathbb{F}}(n, m, d)$ denote the deterministic time complexity of finding a solution $\in \mathbb{F}^n$ to a system of m polynomial equations $\in \mathbb{F}[x_1, \ldots, x_n]$ of total degree d (if one exists).*

We will now mention various known upper bounds on $\widetilde{\mathrm{Sys}}_{\mathbb{F}}(n, m, d)$ and $\mathrm{Sys}_{\mathbb{F}}(n, m, d)$ for various fields. In all these bounds, we have suppressed a $\mathrm{poly}(c)$ dependence in the running time, where $c = \log q$ if $\mathbb{F} = \mathbb{F}_q$ and c is the maximum bit complexity of any coefficient of f if $\mathbb{F}$ is infinite.

Theorem 3.36. *Let $f_1, f_2, \ldots f_m \in \mathbb{F}[x_1, \ldots, x_n]$ be n-variate polynomials of degree at most d. Then, the complexity of finding a single solution to the system $f_1(x) = 0, \ldots, f_m(x) = 0$ (if one exists) over various fields is as follows:*

1. *For all fields $\mathbb{F}$, $\widetilde{\mathrm{Sys}}_{\mathbb{F}}(n, m, d) = \mathrm{poly}((nmd)^{3^n})$. This follows from standard techniques in elimination theory, see [CLO15] for details. For a detailed sketch of the argument and a bound on the size of the extension, see Appendix A.*
2. *[HW99][7] For $\mathbb{F} = \mathbb{F}_q$, $\mathrm{Sys}_{\mathbb{F}}(n, m, d) = O(d^{n^n} \cdot (m \log q^{O(1)}))$.*
3. *[GV88] For $\mathbb{F} = \mathbb{R}$, $\mathrm{Sys}_{\mathbb{F}}(n, m, d) =$ Det-$\mathrm{Sys}_{\mathbb{F}}(n, m, d) = \mathrm{poly}((md)^{n^2})$. Note that in this case the assumption is that the coefficients are integers or rationals[8]. However, the output might be a tuple*

[7] the main results of this work is written for the case when q is prime, but the authors observe that it works for general q as well.

[8] Here the authors assumed that the constants appearing in the system are integers (or rationals). Note that for all computational applications we can WLOG assume this by simply approximating/truncating a given real number at some number of bits.

of algebraic numbers over $\mathbb{R}$ where the degree of the extension is polynomially bounded when n is a constant. See [GV88] for details. Note that all algebraic algorithms used in this paper will continue to hold when the inputs are algebraic numbers of low/polynomial degree, and we deal with algebraic extensions in the standard way.

4. *[Ier89] For $\mathbb{F} = \mathbb{C}$ (or any algebraically closed field), $\mathrm{Sys}_{\mathbb{F}}(n, m, d) = Det\text{-}\mathrm{Sys}_{\mathbb{F}}(n, m, d) = (mn)^{O(n)} \cdot d^{O(n^2)}$.*

Note that, for all cases described above, both $\mathrm{Sys}_{\mathbb{F}}(n, m, d)$ and $\widetilde{\mathrm{Sys}}_{\mathbb{F}}(n, m, d)$ are bounded by $\mathrm{poly}((nmd)^{n^n})$. Thus, when $n = O(1)$, $\mathrm{Sys}(n, m, d) = \mathrm{poly}(m, d)$.

For clarity in presentation, we *artificially* define $\mathrm{Sys}(n, m, d)$ as the complexity of finding a solution to a system of m polynomial equations $\in \mathbb{F}[x_1, \ldots, x_n]$ of total degree d s.t. the solution has to lie in $\mathbb{F}$ if $\mathbb{F} = \mathbb{R}, \mathbb{C}, \mathbb{F}_q$ or an algebraically closed field, and it could be over an algebraic extension for other fields. Clearly, as discussed above $\mathrm{Sys}(n, m, d) = \mathrm{poly}((nmd)^{n^n})$.

3.8.1 Derandomizing solving system of equations:

Derandomizing solving system of equation in general is considered a hard problem for the following reason. Just solving a univarite quadratic equation over $\mathbb{F}_p$ in *deterministic* $\mathrm{poly}(\log p)$ time is a notoriously hard open problem, See [AM94, Problem 15]. Interestingly, this is the only case when low-variate plynomial system solving is hard to derandomize. That is, if the underlying field is not a finite field with large characteristic, then there do exist efficient deterministic algorithms for low-variate system solving.

Indeed solving systems of polynomial equations is the only place in the paper where randomness is utilized. Thus, all our algorithms can be derandomized over $\mathbb{R}, \mathbb{C}$, since the algorithms mentioned in Theorem 3.36, for polynomial system solving over $\mathbb{F} = \mathbb{R}$ and $\mathbb{C}$ are already deterministic. Though we did not mention it, polynomial system solving (and hence our algorithms) can also be derandomized over $\mathbb{F}_{p^d}$(in time $\mathrm{poly}(p, d)$ time).

3.9 Hardness of computing Tensor rank.

The first step towards understanding the computational complexity was by Håstad [Hås90] who showed that determining the tensor rank is an NP-hard over $\mathbb{Q}$ and NP-complete over finite fields. A better way to understand hardness results for computing tensor rank is to study its connection to solving system of polynomial equations.

Theorem 3.37. *[SS16] For any field $\mathbb{F}$, given a system of m algebraic equations S over $\mathbb{F}$, we can in polynomial time construct a 3 dimension tensor $\mathcal{T}_S$ of shape $[3m] \times [3m] \times [n+1]$ and an integer $k = 2m+n$ such that S has a solution $\in \mathbb{F}$ iff $\mathcal{T}$ has rank atmost $2m + n$ over $\mathbb{F}$.*

This shows equivalence between system solving and computing tensor rank. This along with complexity of system solving (discussed in the previous section) shows that computing tensor rank is NP-complete over finite fields, over $\mathbb{R}$ it is in PSPACE [Can88] and is in the Polynomial Hierarchy (Σ_2), assuming the GRH [Koi96].

Similar, reductions also hold for integral domains (e.g. $\mathbb{Z}$) [Shi16], thus showing that computing Tensor rank is *undecidable* over $\mathbb{Z}$ and not known to be decidable over $\mathbb{Q}$. Due to the equivalence between tensor rank computation and learning $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}$ circuits with optimal top fan-in, we get the corresponding hardness consequences for $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}$-circuit reconstruction as well.

Such results also hold for symmetric rank computation, see [Shi16]. Concretely, for 3-dimensional tensors of length n, Shitov showed that we can convert general tensors $\mathcal{T}$ to symmetric tensors $\mathcal{T}_{sym}$ s.t. $rank(\mathcal{T}) + 4.5(n^2 + n) = \text{symmetric-rank}(\mathcal{T}_{sym})$, thus transferring the results mentioned above for general tensors to symmetric tensors as well. Again, these hardness results along with equivalence between symmetric tensor rank computation and reconstructing optimal (w.r.t top fan-in) $\Sigma\wedge\Sigma$ circuits implies that proper learning (with optimal top-fan-in) for $\Sigma\wedge\Sigma$ circuits is as hard as polynomial system solving. In particular, it is NP-hard for most fields and maybe even undecidable over $\mathbb{Q}$.

4 Reconstruction of $\Sigma\wedge\Sigma(k)$ circuits (decomposing low rank symmetric tensors)

In this section we will provide a proof of Theorem 1.4, and we will present our algorithm for reconstructing $\Sigma\wedge\Sigma(k)$ circuits given black-box access to the polynomial computed by it. As discussed in section 3.6, this is equivalent to the problem of finding the optimal symmetric tensor decomposition for low rank symmetric tensors.

In all running times stated in this section, we have suppressed a $\text{poly}(c)$ multiplicative dependence in the running time, where $c = \log q$ if $\mathbb{F} = \mathbb{F}_q$ and c is the maximum bit complexity of any coefficient of f if $\mathbb{F}$ is infinite.

We now restate Theorem 1.4 (only for the randomized algorithm over general fields) and prove it. After the proof we will comment on how the algorithm can be derandomized over $\mathbb{R}$ and $\mathbb{C}$.

Theorem 4.1. *Given black-box access to a degree d polynomial $f \in \mathbb{F}[x_1, x_2, \ldots, x_n]$ such that f is computable by a $\Sigma\wedge\Sigma(k)$ circuit C_f over field $\mathbb{F}$(characteristic $> d$ or 0), there is a randomized $\text{poly}(dk)^{k^{10}}$ time algorithm that outputs a $\Sigma\wedge\Sigma(k)$ circuit computing f.*

Remark 4.2. *when $\mathbb{F} = \mathbb{F}_q$ for $q > nd$ and when $\mathbb{F} = \mathbb{R}$ or $\mathbb{C}$, the output circuit is over the same underlying field $\mathbb{F}$. In general the output circuit might be over an algebraic extension of $\mathbb{F}$.*

Proof. The main observation is that if f can be represented by a $\Sigma\wedge\Sigma(k)$ circuit, then f has only k essential variables. Thus by Lemma 3.23, there is an algorithm that given black-box access to f, runs in time $\text{poly}(n, d)$ and outputs an invertible linear transformation $A \in \mathbb{F}^{n\times n}$ such that $f(A\cdot X)$ depends only on k variables. Let $g_A(X) = f(A \cdot X)$. Since we can compute A, hence given black-box access to f, we can simulate black-box access to g_A in time $\text{poly}(n, d)$.

Notice that g_A is a degree d polynomial in k variables that is also computed by a $\Sigma\wedge\Sigma(k)$ circuit. We will show how to efficiently learn a $\Sigma\wedge\Sigma(k)$ representation of g_A. Since $g_A(A^{-1} \cdot X) = f(X)$, thus given a $\Sigma\wedge\Sigma(k)$ representation of g_A we can obtain a $\Sigma\wedge\Sigma(k)$ representation of f.

The algorithm for learning a $\Sigma\wedge\Sigma(k)$ representation of g_A works as follows. It starts by learning g_A as a sum of monomials (i.e. the sparse polynomial representation of g_A). In particular, let S denote the collection of non-negative integer n-tuples summing to d. The algorithm finds a collection of coefficients $\{c_{\bar{e}} \in \mathbb{F} | \bar{e} \in S\}$ such that $g_A = \sum_{\bar{e}\in S} c_{\bar{e}} \cdot \bar{x}^{\bar{e}}$. This can be done in $\text{poly}(\binom{k+d}{d}) = \text{poly}(d^k)$ using known sparse polynomial reconstruction algorithms [KS01, BOT88].

Let

$$C_{g_A} \equiv \sum_{i=1}^{k} (a_{i,1}x_1 + a_{i,2}x_2 + a_{i,3}x_3 + \ldots + a_{i,k}x_k)^d$$

be the $\Sigma\wedge\Sigma(k)$ circuit computing g_A.

Thus,

$$\sum_{i=1}^{k} (a_{i,1}x_1 + a_{i,2}x_2 + a_{i,3}x_3 + \ldots + a_{i,k}x_k)^d = \sum_{\bar{e}} c_{\bar{e}} \cdot \bar{x}^{\bar{e}}.$$

Now for each monomial $\bar{x}^{\bar{e}}$, $\bar{e} \in S$, we can compare the coefficient of $\bar{x}^{\bar{e}}$ on both sides to get a polynomial equation in the variables $a_{i,j}$. Doing this for all monomials gives us a system of at most $\binom{k+d}{d}$ polynomial equations in k^2 variables, with $a_{i,j}$ as variables. By Theorem 3.36, this system can be solved in time $\text{Sys}(k^2, \binom{k+d}{d}, d)$. Thus, total time complexity is bounded by $\text{poly}(dk)^{k^{10}}$. □

Derandomization: In the above proof, randomness is used in the variable reduction step (Lemma 3.23) and polynomial system solving (Theorem 3.36). Over $\mathbb{R}$ and $\mathbb{C}$, Theorem 3.36 in fact states that polynomial system solving can be done *deterministically* in the same time complexity.

Moreover, for derandomized variable reduction, we can use Lemma 3.24 instead of Lemma 3.23. Observe that all the assumptions of Lemma 3.24 are satisfied, since as f is computed by a $\Sigma\wedge\Sigma(k)$ circuit, it has at most

k essential variables, and the class is also closed under taking first order partial derivatives. Furthermore, $\underbrace{\mathcal{C}+\ldots+\mathcal{C}}_{k\,times}$ for $\mathcal{C}=\Sigma\wedge\Sigma(k)$ is just the class if $\Sigma\wedge\Sigma(k^2)$ circuits, so by Lemma 3.8, there is an efficient hitting set for $\underbrace{\mathcal{C}+\ldots+\mathcal{C}}_{k\,times}$.

Thus, putting it all together we see that we can derandomize the algorithm for proper learning algorithm for $\Sigma\wedge\Sigma(k)$ circuits over $\mathbb{R},\mathbb{C}$.

5 Reconstructing $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}$ circuits (decomposing low rank tensors).

In this section we will provide a proof of Theorem 1.1, and we will present our algorithm for reconstructing $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuits given black-box access to the polynomial computed by it. We will first present all the details for the randomized algorithm and then later comment on how to derandomize it over certain fields.

In all running times stated in this section, we have suppressed a $\mathrm{poly}(c)$ multiplicative dependence in the running time, where $c=\log q$ if $\mathbb{F}=\mathbb{F}_q$ and c is the maximum bit complexity of any coefficient of f if $\mathbb{F}$ is infinite.

Before we prove Theorem 1.1, we develop a bunch of lemmas and subroutines that will be used in the final algorithm.

5.1 Width reduction for $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuits

We first show that there is an algorithm for learning $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuits of arbitrary width w in roughly the same amount of time it takes to learn $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuits of width k.

The algorithm achieves this by a certain "width reduction" procedure that maps the given circuit to one of low width while ensuring we can still get black-box access to it. The algorithm then learns the low width circuit and inverts the map to recover the original possibly high width circuit. A similar "width reduction" technique also appeared in a work of Gupta, Kayal and Lokam [GKL12], but there it was simpler since it was specialized to the case of top fan-in 2, and hence it avoided some of the subtleties that arise here.

Lemma 5.1. *Given black-box access to a degree d, n variate polynomial $f\in\mathbb{F}[X]$ such that f is computable by (arbitary width) $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuit C_f over the field $\mathbb{F}$ with $|\mathbb{F}|>dn$(otherwise we can work with an extension), there is a randomized polynomial-time algorithm that outputs the following:*

1. *Black-box access to another related polynomial h which is computed by a width-k $\Sigma\Pi\Sigma_{\sqcup_j Y_j}(k)$ circuit C_h. Each black-box query to h can be simulated in polynomial time by a suitable related query to f.*
2. *The underlying partition of the Y-variables, which is of the form $Y:=\sqcup_{i=1}^{d} Y_i$ where $Y_j:=\{y_{1,j},y_{2,j},\ldots,y_{k_j,j}\}$. For all $j\in[d]$, $k_j\le k$ thus width$(h)\le k$.*
3. *A collection of linear polynomials $\{P_{i,j}\in\mathbb{F}[X_j]\}_{j\in[d],i\in[k_j]}$ such that the following holds: Upon substituting $y_{i,j}=P_{i,j}$ into the polynomial h (for each each variable $y_{i,j}$ that appears in h), we recover f. Moreover $C_h(y_{i,j}=P_{i,j})$ is a $\Sigma\Pi\Sigma_{\{\sqcup_i X_i\}}(k)$ representation of f.*

Proof. Let $C_f=\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ be the depth-3 set-multilinear circuit representation of f. Let

$$C_f\equiv\sum_{i=1}^{k}\prod_{j=1}^{d}\ell_{i,j}\qquad\text{where } \ell_{i,j} \text{ is a linear polynomial in } \bar{X}_j \text{ variables.}$$

Without loss of generality we will assume that k is the smallest integer such that f has such a representation. The reason we can do this because of the following. If there is a representation of f with a smaller top fan-in k', then we can assume that the algorithm knows k' and runs the algorithm with k' instead of k.

The reason we can assume the algorithm "knows" k' is that we can run the algorithm for all values of k' from 1 up to k and try to learn the circuit with that top fan-in, and the first time it successfully learns a circuit will correspond the representation of f with the lowest top fan-in. The algorithm knows when it has successfully learnt the circuit because it can test whether or not the output circuit agrees with the input f using polynomial identity testing (See Lemma 3.7 and Lemma 3.8).

As a first step, the algorithm will learn the linear span of the set $\{\ell_{j,i} | j \in [k]\}$ for each $i \in [d]$.

To do this, substitute all variables in $X \setminus X_i$ to independent randomly chosen values from $\mathbb{F}$ and interpolate to get a linear polynomial $P_{1,i} \in \mathbb{F}[X_i]$. Then $P_{1,i} = \alpha_{1,i}^{(1)}\ell_{1,i} + \alpha_{2,i}^{(1)}\ell_{2,i} + \ldots \alpha_{k,i}^{(1)}\ell_{k,i}$ where $\alpha_{j,i}^{(1)} \in \mathbb{F}$. Observe that the algorithm learns $P_{1,i}$ but the $\alpha_{j,i}^{(1)}$ and $\ell_{j,i}$ are unknowns. We repeat this procedure for another $k-1$ random independent substitutions of the variables of $X \setminus X_i$ and upon interpolation recover $k-1$ additional linear combinations of $\ell_{1,i}, \cdots, \ell_{k,i}$. Let the resulting learnt linear polynomials be $P_{2,i}, P_{3,1}, \ldots P_{k,i}$ respectively.

Now, we have that,

$$\begin{pmatrix} P_{1,i} \\ P_{2,i} \\ \vdots \\ P_{k,i} \end{pmatrix} = \underbrace{\begin{pmatrix} \alpha_{1,i}^{(1)} & \alpha_{2,i}^{(1)} & \cdots & \alpha_{k,i}^{(1)} \\ \alpha_{1,i}^{(2)} & \alpha_{2,i}^{(2)} & \cdots & \alpha_{k,i}^{(2)} \\ \vdots & \vdots & \vdots & \vdots \\ \alpha_{1,i}^{(k)} & \alpha_{2,i}^{(k)} & \cdots & \alpha_{k,i}^{(k)} \end{pmatrix}}_{A_{\bar{\alpha_i}}} \cdot \begin{pmatrix} l_{1,i} \\ l_{2,i} \\ \vdots \\ l_{k,i} \end{pmatrix} \tag{4}$$

Now the great advantage of our assumption that k is the smallest integer such that f has a $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ representation is that we can invoke Lemma 3.21. In conjunction with Lemma 3.16, this implies that $A_{\bar{\alpha_i}}$ is invertible with high probability.

Thus note that for each $j \in [k]$, $\ell_{j,i}$ is in the linear span of $\{P_{1,i}, \ldots, P_{k,i}\}$.

If $\{P_{1,i}, P_{2,i}, \ldots, P_{k,i}\}$ are linearly independent polynomials, then the algorithm does the following. It introduces k new formal variables $y_{1,i}, \ldots, y_{k,i}$ and defines k linear functions in these variables as follows. For each $j \in [k]$, it defines $\tilde{l}_{j,i}$ by the following equation.

$$\begin{pmatrix} \tilde{l}_{1,i} \\ \tilde{l}_{2,i} \\ \vdots \\ \tilde{l}_{k,i} \end{pmatrix} = A_{\bar{\alpha_i}}^{-1} \begin{pmatrix} y_{1,i} \\ y_{2,i} \\ \vdots \\ y_{k,i} \end{pmatrix} \tag{5}$$

If $\{P_{1,i}, \ldots, P_{k,i}\}$ are not linearly independent then find a set $S_i \subseteq [k]$ such that the elements of $B_i = \{P_{j,i} | j \in S_i\}$ form a basis of $\{P_{1,i}, \ldots, P_{k,i}\}$. For instance, one can find S_i using Lemma 3.17. let $|S_i| = k_i$.

In this case, change Equation 5 by keeping $y_{j,i}$ for $j \in S_i$ as a formal variables and replacing $y_{j,i}$ for each $j \notin S_i$ in the following way: If $P_{j,i} = \sum_{j \in S_i} \lambda_j P_{j,i}$, then replace $y_{j,i} = \sum_{j \in S_i} \lambda_j y_{j,i}$. Observe that in both cases the width of $\tilde{l}_{j,i}$ is $\leq k$.

The algorithm performs the above procedure for each $i \in [d]$, and thus for each $i \in [d]$ and each $j \in [k]$ it obtains a linear polynomial $\tilde{l}_{j,i}$.

Now consider the following polynomial $h = \sum_{j=1}^{k} \prod_{i=1}^{d} \tilde{l}_{j,i}$. Notice that h is a $\Sigma\Pi\Sigma_{\{\sqcup_i Y_i\}}(k)$ circuit of width at most k, with underlying partition $Y := \sqcup_{i=1}^{d} Y_i$, where $Y_i := \{y_{1,i}, y_{2,i}, \ldots, y_{k_i,i}\}$, $k_i = |S_i|$.

Claim 5.2. *There is an efficient polynomial-time algorithm for simulating black-box access to h.*

Proof. Suppose the algorithm wants to evaluate h at an input β, where $\beta = (\beta_{j,i})_{i \in [d], j \in k_i}$, $\beta_{j,i} \in \mathbb{F}$.

This can be done by solving the following system of linear equations in the X variables, to obtain a solution X_0, and then evaluating f at X_0. Suppose that for each $i \in [d]$, $S_i := \{i_1, i_2, \ldots, i_{k_i}\}$. The for each $i \in [d]$, add the following equations to the system of equations.

$$\begin{pmatrix} P_{i_1,i}(X_i) \\ P_{i_2,i}(X_i) \\ \vdots \\ P_{i_{k_i},i}(X_i) \end{pmatrix} = \begin{pmatrix} \beta_{i_1,i} \\ \beta_{i_2,i} \\ \vdots \\ \beta_{i_{k_i},i} \end{pmatrix}$$

Existence of a solution is guaranteed, since for each $i \in [d]$, $\{P_{j,i}(X_i) | j \in S_i\}$ is a linearly independent collection of linear polynomials, and hence a solution can be efficiently found by Gaussian elimination. □

By Equations 4 and Equation 5, we conclude that upon substituting $y_{i,j} = P_{i,j}$ in h we recover f. Moreover, $C_h(y_{i,j} = P_{i,j})$ is a $\Sigma\Pi\Sigma_{\{\sqcup_i X_i\}}(k)$ representation of f.

□

Corollary 5.3. *Suppose A is an algorithm that has the following behavior. On input black-box access to degree d, n-variate polynomial $f \in \mathbb{F}[X]$ such that f is computable by a width k $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuit C_f over the field $\mathbb{F}$, runs in randomized time $\mathcal{A}(n,d,k)$ and outputs a $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuit computing f. Then there is another algorithm A' that has the following behavior. On input black-box access to degree d, n-variate polynomial $f \in \mathbb{F}[X]$ such that f is computable by an arbitrary width $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuit C_f over the field $\mathbb{F}$, runs in randomized time $\mathrm{poly}(n,d,k) \cdot \mathcal{A}(n,d,k)$ and outputs a $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuit computing f.*

Proof. The algorithm A' works as follows. Using the procedure described in Lemma 5.1, it uses black-box queries to f to simulate black-box queries to another polynomial h which is computed by a degree d, width k, $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuit in at most n variables. It then uses algorithm A to obtain a $\Sigma\Pi\Sigma_{\sqcup_j Y_j}(k)$ representation of h in time $\mathrm{poly}(n,d,k) \cdot \mathcal{A}(n,d,k)$ (since each query to h takes time $\mathrm{poly}(n,d,k)$) and then makes the suitable substitution of linear polynomials into the variables of h to recover f. □

5.2 Reconstructing low degree $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuits

As a basic step in reconstructing general $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuits, we show how to reconstruct $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuits efficiently when the degree of the computed polynomial (which corresponds to the dimension of the underlying tensor) is small.

We will obtain a bound as a function of the width-w, but when we use the lemma later, we will assume the width is k (due to our width reduction lemma). (Recall, the width w is an upper bound on the number of variables in each X_i.) The notation we use in the running time of the lemma below is from Section 3.8.

Lemma 5.4. *Given black-box access to a degree d polynomial $f \in \mathbb{F}[X]$ such that f is computable by a width w $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuit C_f over the field $\mathbb{F}$, there is a randomized $\mathrm{Sys}(kwd, w^d, d) + \mathrm{poly}(w,k,d)$ time algorithm that outputs a $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuit computing f.*

Remark 5.5. *when $\mathbb{F} = \mathbb{F}_q$ and when $\mathbb{F} = \mathbb{R}$ or $\mathbb{C}$, the output circuit is over the same underlying field $\mathbb{F}$. In general the output circuit might be over an algebraic extension of $\mathbb{F}$.*

Proof. Start by learning f as a sparse polynomial, that is find $c_{\bar{e}} \in \mathbb{F}$ such that $f = \sum_{\bar{e} \in X_1 \times \cdots \times X_d} c_{\bar{e}} \cdot \bar{x}^{\bar{e}}$. This can be done in $\mathrm{poly}(n,d)$ using [KS01, BOT88] sparse polynomial reconstruction.

Also, let

$$C_f \equiv \sum_{i=1}^{k} \prod_{j=1}^{d} (a_{i,j,1}x_{j,1} + a_{i,j,1}x_{j,1} + a_{i,j,2}x_{j,2} + \ldots + a_{i,j,1}x_{j,1})$$

be a $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$circuit computing f.

Thus,

$$\sum_{i=1}^{k} \prod_{j=1}^{d} \sum_{\ell=1}^{w} a_{i,j,\ell} \cdot x_{j,\ell} = \sum_{\bar{e} \in X_1 \times \cdots \times X_d} c_{\bar{e}} \cdot \bar{x}^{\bar{e}}$$

gives us a system of polynomial equations with $a_{i,j,\ell}$ as variables. Notice that, this system has w^d polynomial equations of degree d supported on kwd variables. Thus, can be solved in $\mathrm{Sys}_{\mathbb{F}}(kwd, w^d, d)$. Since, system solving is the most expensive step of this process thus the overall time complexity is bounded by $\mathrm{Sys}(kwd, w^d, d)\mathrm{poly}(w, k, d) = \mathrm{poly}(kwd \cdot w^d \cdot d \cdot c)^{(kwd)^{(kwd)}}$. □

5.3 Learning 2 linear forms appearing in the circuit

As a first step towards learning a general (high degree) $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuit C, our algorithm will try to learn a single linear form appearing in the circuit C. In fact it will be convenient to learn 2 linear forms appearing in C such that each multiplication gate of C contains at most one of them.

Lemma 5.6. *Let $C \equiv \sum_{i=1}^{k} T_i$ be a width-k, simple and minimal $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuit computing a nonzero polynomial f, such that $k \geq 2$ and $d > 2k^2$. Then, given black-box access to f, there is a randomized algorithm that runs in time $\mathrm{Sys}(2k^4, k^{2k^2}, 2k^2) + \mathrm{poly}(k, d)$, and does the following. It outputs a set L of $2k^3$ pairs of linear forms which has the following property. One of the pairs (ℓ_1, ℓ_2) in L is such that for some $i \in [d]$, ℓ_1 and ℓ_2 are supported on the variables of X_i, and both ℓ_1 and ℓ_2 appear in C.*

Proof. Let $\sigma = \sqcup_{2k^2+1}^{d} \bar{\alpha}_i$, be a random assignment from the underlying field $\mathbb{F}$ to the variables in $\sqcup_{i=2k^2+1}^{d} X_i$ performed in the following way: we pick $S \subseteq \mathbb{F}$ such that $|S| > 2^{k+1}nd$. If $\mathbb{F}$ is not large enough, then we can pick S from an extension field. Then each element of $\sqcup_{i=2k^2+1}^{d} X_i$ is independently set to a uniformly random element of S, and let the resulting polynomial be $f|_\sigma$.

Then observe that after setting these variables, with high probability, the restricted circuit $C|_\sigma$ is still nonzero, simple and minimal. Simplicity follows directly. To observe non-zeroness and minimality, consider the following family of polynomials, $\forall U \subset [k], f_U := \sum_{i \in U} T_i$. Note that, using minimality of f, we get that $\forall U \subset [k]$, $f_U \neq 0$, thus on applying Lemma 3.7 on $\prod_{\forall U} f_U$ gives that for aforementioned random σ, $f_U|_\sigma \neq 0$ implying non-zeroness and minimality of $C|_\sigma$.

Now we use Lemma 5.4 to learn a $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuit with variable partition $\sqcup_{i=1}^{2k^2} X_i$ computing $f|_\sigma$. Let it be $\tilde{C} = \sum_{i=1}^{k'} B_i$, with $k' \leq k$ and WLOG $\tilde{C}$ is minimal (else we remove all identically zero subcircuits).

The algorithm then outputs a set L which comprises of all pairs of linear forms appearing in $\tilde{C}$ that are supported on the same set of variables X_i (for each X_i). Let us now see why this set L has the desired property.

By Lemma 3.14, we get that for each $T_i|_\sigma$ there exists $j_i \in [k]$ such that $\Delta(T_i|_\sigma, B_{j_i}) \leq k$. Let us call the set of X_i's on which $\frac{T_i|_\sigma}{\gcd(T_i|_\sigma, B_{j_i})}$ is supported "bad". Then, the number of sets that are bad for at least one choice of T_i is bounded by k^2. Thus, since $d > 2k^2$, there exists at least one set (say X_a) such that if a linear form $\ell \in \mathbb{F}[X_a]$ is such that ℓ divided $T_i|_\sigma$, then ℓ also divides B_{j_i}. Also, since $C|_\sigma$ is simple, there are at least two distinct linear forms $\ell_1, \ell_2 \in \mathbb{F}[X_a]$ that each divide some multiplication gate of $C|_\sigma$. Clearly this pair of linear forms is also included in L. The time complexity estimate is immediate. □

5.4 Learning most of the linear forms appearing in the circuit

In this section we will see how to use the two linear forms learnt in the previous subsection to learn a small set S of multiplication gates, such that each multiplication gate T_i of C is very "close" to some gate of S. From this we will then see how to essentially learn *most* of the linear forms appearing in each gate of C. Our approach is recursive, so we will assume using an induction hypothesis that there is $\mathcal{A}(n, k-1, d)$ time randomized algorithm for reconstructing degree d, width k $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k-1)$ circuits. Note that, for base case of $k = 1$ follows directly from black-box factoring result of [KT90] i.e. $\mathcal{A}(n, 1, d) = \mathrm{poly}(n, d)$.

Lemma 5.7. *Let $C \equiv \sum_{i=1}^{k} T_i$ be a minimal $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuit of degree d and let ℓ_1 and ℓ_2 be two distinct linear forms supported on variables of X_i for some $i \in [d]$ such that each of ℓ_1 and ℓ_2 appears in C. Then there is a randomized algorithm that given black-box access to C, given ℓ_1 and ℓ_2, and two oracle calls to algorithm for learning $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k-1)$ circuits, runs in time atmost $2\mathcal{A}(n, k-1, d) + \mathrm{poly}(n, k, d)$*

and outputs a set $S = \{M_1, M_2 \ldots, M_{|S|}\}$ of at most $2k-2$ $\Pi\Sigma$ circuits of degree $d-1$ such that with high probability, for all $i \in [k]$, there exists $j \in [2k-2]$ such that $\Delta(T_i, M_j) < 2k$.

Proof. Let $\bar{\alpha}_1$ be a setting of the variables of X_i (chosen from a suitably large domain) on which ℓ_1 vanishes. Notice that with high probability, ℓ_1 is the only linear form appearing in C that vanishes, and the restricted circuit $C|_{X_i=\bar{\alpha}_i}$ is a $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k-1)$ circuit. Indeed it is also possible that it might have much few than $k-1$ gates. Using the induction hypothesis, there is an efficient algorithm to learn a $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k-1)$ representation of $f|_{X_i=\bar{\alpha}_1}$. We run this algorithm and let the output be C_1.

Similarly we let $\bar{\alpha}_2$ be a setting of the variables of X_i (chosen from a suitably large domain) on which ℓ_2 vanishes. We learn a $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k-1)$ representation of $f|_{X_i=\bar{\alpha}_2}$ and let the output be C_2.

Let S denote the set of union multiplication gate from C_1 and C_2. Note, $|S| \leq 2(k-1)$. Observe that for each $i \in [k]$, with high probability, one of the substitutions $X_i = \bar{\alpha}_1$ of $X_i = \bar{\alpha}_2$ must keep T_i nonzero, since at most one of ℓ_1, ℓ_2 divides T_i. Thus, by Lemma 3.14, for all $i \in [k]$, T_i must be "close" to some multiplication gate of C_1 or C_2. More precisely, there exists $j \in [2k-2]$ such that $\Delta(T_i, M_j) < 2k$. The time complexity estimates is immediate. □

Thus we can now assume that our algorithm can compute a set S consisting of multiplication gates such that $|S| \leq 2(k-1)$, and each gate of the circuit C that we are trying to learn is close to some element of S.

Lemma 5.8. *Let $C \equiv \sum_{i=1}^{k} T_i$ be a $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuit of degree d and let $S = \{M_1, M_2 \ldots, M_{|S|}\}$ be a set of at most $2k-2$ $\Pi\Sigma$ circuits of degree $d-1$ such that for all $i \in [k]$, there exists $j \in [2k-2]$ such that $\Delta(T_i, M_j) < 2k$. Then there is a $\mathrm{poly}((kd^{k^3}))$-time algorithm for computing another set $\bar{S}$ which has the following properties.*

1. $|\bar{S}| \leq \left(|S| \cdot \binom{d-1}{2k^2}\right)^k$
2. *The elements of $\bar{S}$ are k-tuples of $\Pi\Sigma$ circuits of degree $d-1-2k^2$*
3. *One of the elements of $\bar{S}$ is of the form $(G_1, G_2, \ldots G_k)$ where for each $i \in [k]$, G_i divides T_i. Moreover all of the G_is are set-multilinear $\Pi\Sigma$ circuits sharing the same variable partition.*

Proof. Consider the set $\hat{S} = \{G \mid \exists M \in S \text{ s.t. } G \text{ divides } M, \text{ and } \deg(G) = d-1-2k\}$. Notice that, $|\hat{S}| \leq |S| \cdot \binom{d-1}{2k}$.

By the property of the set S, this implies that for all $i \in [k]$, there is some element $G_i' \in \hat{S}$ such that G_i' divides T_i. There may be multiple elements of $\hat{S}$ that divide T_i, but we fix any one and call it G_i'. Now consider the set $\{G_1', G_2', \ldots, G_k'\}$.

Ideally we would like these G_i' to depend on the same set of variables. Here is a modification of them that will result in somewhat lower degree polynomials, but they would be supported on the same variable partition. First recall that all the G_i's are a product of set disjoint linear forms, and each linear form is supported on one of the parts of the underlying partition $X = \sqcup_{i=1}^{d} X_i$. Now we perform the following procedure. For each $i \in [k]$ and for each $j \in [d]$ such that G_i' does not have a factor $\in \mathbb{F}[X_j]$ then remove(divide out) any linear factor $\in \mathbb{F}[X_j]$ from all the other elements of the set $\{G_1', G_2', \ldots, G_k'\}$. At the end of this process, the new polynomials have degree at least $d-1-2k^2$ and they all are supported on the same parts of the partition $X = \sqcup_{i=1}^{d} X_i$. We can also ensure that they all have degree exactly $d-1-2k^2$, as we can divide out linear forms all depending on the same set of variables from all of these polynomials till they have degree exactly $d-1-2k^2$. Call these new polynomials $G_1, G_2, \ldots, G_k$. Observe that they all have the same variable partition, they all have degree $d-1-2k^2$, and they each divide some multiplication gate T_i of the circuit C as well as some element of S.

We will now define the set $\bar{S}$. First consider the set $Q = \{G | \exists M \in S \text{ s.t. } G \text{ divides } M, \text{ and } \deg(G) = d-1-2k^2\}$. Let $\bar{S}$ be the set of all k-tuples of elements of Q. Then observe that $(G_1', G_2', \ldots, G_k')$ is an element of $\bar{S}$. Then $|\bar{S}| \leq \left(|S| \cdot \binom{d-1}{2k^2}\right)^k$ and it satisfies all other required properties as well.

□

5.5 Learning the full circuit

Lemma 5.9. *Let $C \equiv \sum_{i=1}^{k} T_i$ be a $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuit of degree d and width w computing a polynomial f. Let $(G_1, G_2, \ldots G_k)$ be a k-tuple where for each $i \in [k]$, G_i divides T_i. Moreover all of the G_is are set-multilinear $\Pi\Sigma$ circuits of degree $d-1-2k^2$, sharing the same variable partition. Then, given black-box access to C and given the k-tuple $(G_1, G_2, \ldots G_k)$, there is a randomized $\mathrm{Sys}(k(2k^2+1)w, (2k^2w)^{2k^2+1}, 2k^2+1) + \mathrm{poly}((2k^2w)^{2k^2})$ time algorithm that outputs a $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ representation of f.*

Proof. We are given $(G_1, G_2, \ldots G_k)$ and our aim is to find H_i-s such that $f = \sum_{i\in[k]} G_i H_i$ is a $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ representation of f. Notice that, if we have black-box access to the individual H_i-s, then we can learn them just by black-box factorization followed by sparse reconstruction of the linear factors. We will achieve something close to this in principle.

As a first step, we find the linear dependency structure among the G_i-s using Lemma 3.17.

Let $G_1, G_2, \ldots G_c$ be a basis of linear space of G_i-s (we can always ensure this by relabelling of gates). Also, let M be a $k \times c$ matrix which is the corresponding linear dependence matrix we get from Lemma 3.17, that is,

$$M_{k\times c} \begin{pmatrix} G_1 \\ \vdots \\ G_c \end{pmatrix} = \begin{pmatrix} G_1 \\ \vdots \\ G_c \\ \vdots \\ G_k \end{pmatrix}. \tag{6}$$

Note that,

$$f = \begin{pmatrix} H_1, \cdots, H_k \end{pmatrix} \cdot \begin{pmatrix} G_1 \\ \vdots \\ G_k \end{pmatrix} = \begin{pmatrix} H_1, \cdots, H_k \end{pmatrix} \cdot M \cdot \begin{pmatrix} G_1 \\ \vdots \\ G_c \end{pmatrix}$$

For $i \in [c]$, define $\tilde{H}_i$ by the following equality

$$\begin{pmatrix} \tilde{H}_1 \\ \vdots \\ \tilde{H}_c \end{pmatrix} := M^T \begin{pmatrix} H_1 \\ \vdots \\ . \\ H_k \end{pmatrix}. \tag{7}$$

Thus,

$$f = \sum_{i\in[k]} H_i G_i = \sum_{i\in[c]} \tilde{H}_i G_i.$$

We will now show how to obtain black-box access to the $\tilde{H}_i$ and then later use the $\tilde{H}_i$ to find the H_i-s.

Now observe that $f = \sum_{i=1}^{c} \tilde{H}_i G_i$ where $G_1, \ldots G_c$ are linearly independent (and we know what $G_1, \ldots G_c$ are). Let $Z \subset X$ be the set of variables on which the G_i's are supported. Then for each i, the $\tilde{H}_i$ and the H_i belong to $\mathbb{F}[X \setminus Z]$.

We will now show how to get get black-box access to the $\tilde{H}_i$ by using black-box access to f and to the G_i-s.

Let $\beta_1, \beta_2, \ldots, \beta_c$ be random independent substitution of Z variables. Let $f_{\beta_i} \in \mathbb{F}[X\setminus Z]$ be the polynomial obtained by substituting β_i into the Z variables of f. Then f_{β_i} is a $(2k^2+1)w$ variate multilinear polynomial (think of w being small, such as k) and thus f_{β_i} can be represented efficiently as a sparse polynomial (and in fact we can learn the monomial representation using black-box sparse polynomial interpolation in $\mathrm{poly}((2k^2w)^{2k^2})$ [KS01]) .

Now observe that

$$\begin{pmatrix} f_{\beta_1} \\ f_{\beta_2} \\ \vdots \\ f_{\beta_c} \end{pmatrix} = \underbrace{\begin{pmatrix} G_1(\beta_1) & G_2(\beta_1) & \cdots & G_c(\beta_1) \\ G_1(\beta_2) & G_2(\beta_2) & \cdots & G_c(\beta_2) \\ \vdots & \vdots & \vdots & \vdots \\ G_1(\beta_c) & G_2(\beta_c) & \cdots & G_c(\beta_e) \end{pmatrix}}_{B_\beta} \cdot \begin{pmatrix} \tilde{H}_1 \\ \tilde{H}_2 \\ \vdots \\ \tilde{H}_c \end{pmatrix}.$$

Since $G_1, \ldots G_c$ are linearly independent, by Lemma 3.16 we see that B_β is invertible with high probability. Thus,

$$\begin{pmatrix} \tilde{H}_1 \\ \tilde{H}_2 \\ \vdots \\ \tilde{H}_c \end{pmatrix} = B_\beta^{-1} \begin{pmatrix} f_{\beta_1} \\ f_{\beta_2} \\ \vdots \\ f_{\beta_c} \end{pmatrix}$$

and in this manner we obtain access to the $\tilde{H}_i$-s. Indeed we can recover the monomial representation of the $\tilde{H}_i$-s.

Note that each $\tilde{H}_i$ is a polynomial is at most $(2k^2+1)w$ variables and is of degree $\leq 2k^2+1$.

For ease of presentation, assume each $H_i \in \mathbb{F}[X_1, X_2, \ldots, X_{2k^2+1}]$, which can ensured by relabeling of variables.

Recall,

$$M^T \cdot \begin{pmatrix} H_1 \\ \cdot \\ \vdots \\ H_k \end{pmatrix} = \begin{pmatrix} \tilde{H}_1 \\ \vdots \\ \tilde{H}_c \end{pmatrix} \tag{8}$$

and let $H_i = \prod_{j \in [2k^2+1]} (a_{i,j,1}x_{j,1} + a_{i,j,2}x_{j,2} \ldots a_{i,j,w}x_{j,w})$, where $a_{i,j,k}$ are unknowns that we intend to find. On expanding equation 8 and comparing coefficient of monomials on both sides, we will get at most $(2k^2w)^{2k^2+1}$ polynomial equations of degree at most $2k^2+1$ in at most $k(2k^2+1)w$ variables. Thus we can solve this in $\mathrm{Sys}(k(2k^2+1)w, (2k^2w)^{2k^2+1}, 2k^2+1)$ time by Theorem 3.36.

□

5.6 Putting it all together

We now show how to combine all the lemmas and subroutines developed so far to get the full reconstruction algorithm for $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuits. The theorem below is basically a restatement of Theorem 1.1 ((only for the randomized algorithm over general fields). After the proof we will comment on how the algorithm can be derandomized over $\mathbb{R}$ and $\mathbb{C}$.

Theorem 5.10. *Given black-box access to a degree d, n-variate polynomial $f \in \mathbb{F}[X]$ such that f is computable by a $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuit C_f over the field $\mathbb{F}$, there is a randomized* $\mathrm{poly}(d^{k^3}, k^{k^{k^{10}}}, n)$ *time algorithm that outputs a $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuit computing f.*

Remark 5.11. *when $\mathbb{F} = \mathbb{F}_q$ and when $\mathbb{F} = \mathbb{R}$ or $\mathbb{C}$, the output circuit is over the same underlying field $\mathbb{F}$. In general the output circuit might be over an algebraic extension of $\mathbb{F}$.*

Proof. By Corollary 5.3, it suffices to assume that the width of f is at most k.

Our algorithm is recursive and we assume that we have an efficient algorithm for reconstructing $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k-1)$ circuits that runs in time $\mathcal{A}(n, k-1, d)$. For the base case of $k = 1$, the reconstruction algorithm follows directly from black-box factoring result of [KT90].

Now assume $k \geq 2$. By Corollary 3.10 we can assume that C_f is simple. We can also assume that C_f is minimal. This is because if f has a $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ representation with a non-minimal circuit, then it also

has $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ representation with a minimal circuit, which is obtained by just deleting any subset of multiplication gates in the non-minimal circuit which sums to zero.

If $d \leq 2k^2$, then we invoke the algorithm in Lemma 5.4 to learn a $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ representation of C.

If $d > 2k^2$, we invoke the algorithm from Lemma 5.6 to compute the set L of pairs of linear forms. For each pair $(\ell_1, \ell_2) \in L$ we do the following: We invoke Lemma 5.7 to compute a set S of at most $2k-2$ $\Pi\Sigma$ circuits and then invoke Lemma 5.8 to compute a set $\bar{S}$ of k-tuples. Let us call this final set $\bar{S}_{(\ell_1,\ell_2)}$. For each k-tuple $(G_1, G_2, \ldots, G_k) \in \bar{S}_{(\ell_1,\ell_2)}$ we invoke the algorithm of Lemma 5.9 with $w = k$ to output a circuit. We then verify that the output circuit indeed has the $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ format and then we check (by running a polynomial identity testing algorithm) if it computes f. If it passes both theses verification steps then the algorithm halts and outputs that circuit. By Lemmas 5.6, 5.7, 5.8 and 5.9, we do know that for some choice of (ℓ_1, ℓ_2) and for some choice of $(G_1, G_2, \ldots, G_k) \in \bar{S}_{(\ell_1,\ell_2)}$, the algorithm will succeed with high probability.

Time complexity analysis: The upper bound on the time complexity is $poly(n, d, k) \cdot \mathcal{A}(n, k, d)$. Recall, $\mathcal{A}(n, k, d)$ is the time complexity of learning degree d, width k $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuit computing f. We now upper bound $\mathcal{A}(n, k, d)$. Note that,

$$\begin{aligned}\mathcal{A}(n,k,d) &\leq 2 \cdot \mathcal{A}(n,k-1,d) + \mathrm{Sys}(k(2k^2+1)k, (2k^2k)^{2k^2+1}, 2k^2+1) + \mathrm{poly}(n,d) + \mathrm{poly}(d^{k^3}).\\ &\leq 2^k \cdot \mathcal{A}(n,1,d) + k \cdot \mathrm{Sys}(k(2k^2+1)k, (2k^2k)^{2k^2+1}, 2k^2+1) + k\mathrm{poly}(d^{k^3}) \leq \mathrm{poly}(d^{k^3}, k^{k^{k^{10}}}, n).\end{aligned}$$

So, the total time complexity is also bounded by $\mathrm{poly}(d^{k^3}, k^{k^{k^{10}}}, n)$. □

Derandomization: We list below the places in the proof where randomization is used, and state how to derandomize them.

- *Polynomial system solving:* This is used in two different places in the proof and in both cases can be substituted with deterministic algorithms for the same when the underlying field is $\mathbb{R}$ or $\mathbb{C}$ (See Theorem 3.36). It is worth noting that this is the only step where the derandomization does not work over all fields.
- *Blackbox factoring:* For this step we had used the randomized blackbox factoring algorithm by Kaltofen and Trager. To derandmize this step one can use the deterministic factoring algorithm for multilinear polynomials given in [SV10] along with a hitting set of $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuits.
- *Variable or "width" reduction:* In this step we had used Lemma 3.16 to find an assignment s.t. A_{α_i} matrix is invertible. However in our setting we can use the deterministic version of this lemma instead, i.e. Lemma 3.18 since we have efficient hitting sets for $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuits and for sums of constantly many $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuits.
- *Computing linear dependence among the G_is:* In this step, instead of using the randomized algorithm from Lemma 3.17), we can instead use the derandomized version of it, i.e. Lemma 3.19 since we have efficient hitting sets for $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuits and for sums of constantly many $\Sigma\Pi\Sigma_{\{\sqcup_j X_j\}}(k)$ circuits.

6 Multilinear Depth-3 Circuits

In this section, we will provide a proof of Theorem 1.6. In all running times stated in this section, we have suppressed a $\mathrm{poly}(c)$ multiplicative dependence in the running time, where $c = \log q$ if $\mathbb{F} = \mathbb{F}_q$ and c is the maximum bit complexity of any coefficient of f if $\mathbb{F}$ is infinite.

Let us start by revisiting some core definition related to depth-3 circuits.

Definition 6.1. *A depth-3 $\Sigma\Pi\Sigma(k)$ circuit C of degree (at most) d computes a polynomial of the form*

$$C \equiv \sum_{i=1}^{k} T_i(X) = \sum_{i=1}^{k} \prod_{j=1}^{d_i} \ell_{i,j}(X),$$

where $d_i \le d$ and the $\ell_{i,j}$-s are linear functions; $\ell_{i,j}(X) = \sum_{t=1}^{n} a_{i,j}^t x_t + a_{i,j}^0$ with $a_{i,j}^t \in \mathbb{F}$.
A multilinear $\Sigma\Pi\Sigma(k)$ *circuit is a $\Sigma\Pi\Sigma(k)$ circuit in which each T_i is a multilinear polynomial. In particular, each such T_i is a product of variable-disjoint linear functions.*

Following notations will be useful throughout the paper.

1. *For each $A \subseteq [k]$, C_A is defined as a subcircuit of C supported on A, formally, $C_A = \sum_{i \in A} T_i$.*
2. $\gcd(C) \triangleq \gcd(T_1, T_2, \ldots, T_k)$.
3. $\mathrm{rank}(C) = \dim(\mathrm{span}\{\ell_{i,j}\})$.

Observe that for a multilinear circuit: $d \le \mathrm{rank}(C)$.

Based on the notion of rank, in [KS09a], Karnin and Shpilka defined a "distance function" for depth-3 circuits.

Definition 6.2 ([KS09a])**.** *For two $\Sigma\Pi\Sigma$ circuits C_1, C_2, we define a* distance *function:*

$$\Delta_{\mathrm{rank}}(C_1, C_2) \triangleq \mathrm{rank}\left(\frac{C_1 + C_2}{\gcd(C_1 + C_2)}\right).$$

For a single circuit C, we define the GCD-free-*rank as:*

$$\Delta_{\mathrm{rank}}(C) \triangleq \Delta_{\mathrm{rank}}(C, 0) = \mathrm{rank}\left(\frac{C}{\gcd(C)}\right).$$

The following result known as the *Rank Bound* provides a structural property for multilinear depth-3 computing the zero polynomial, under some technical conditions.

Theorem 6.3 ([SS09])**.** *There exists a monotone function $R_M(k) \le \mathcal{O}(k^3 \log k)$ such that any simple and minimal, multilinear $\Sigma\Pi\Sigma(k)$ circuit C, computing the zero polynomial satisfies $rank(C) \le R_M(k)$.*

6.1 Learning Low-Degree Multilinear $\Sigma\Pi\Sigma(k)$ Circuits

In this section we will show how to reconstruct a low-degree multilinear $\Sigma\Pi\Sigma(k)$ circuit from black-box samples. (For now we only state the randomized version and later point out how to derandomize it over $\mathbb{R}$ and $\mathbb{C}$). We state the lemma below for general k and d, but think of k and d to be constants, and the number of variables, n, to be growing.

Lemma 6.4. *Let $f \in \mathbb{F}[x_1, x_2, \ldots, x_n]$ be a polynomial computed by a degree d, multilinear $\Sigma\Pi\Sigma(k)$ circuit C_f of the form*

$$\sum_{i=1}^{k} T_i(X) = \sum_{i=1}^{k} \prod_{j=1}^{d_i} \ell_{i,j}(X)$$

Then there is a randomized algorithm that given k, d and black-box access to f outputs a multilinear $\Sigma\Pi\Sigma(k)$ circuit computing f, in time $\mathrm{poly}(n, \mathrm{Sys}(d^2k^2, kd^2n + \binom{dk+d}{k}, d)) \le (dkn)^{(d^2k^3)^{\mathcal{O}(d^2k^2)}}$.

Proof. Let m be the number of essential variables in f. Since there at most kd linear forms appearing in C, this it is easy to see that $m \le kd$.

By Lemma 3.23, there is a polynomial-time randomized algorithm that given black-box access to f, computes an invertible linear transformation $A \in \mathbb{F}^{n \times n}$ such that $f(A \cdot \bar{x})$ only depends on the first m variables.

Let $g(X) = f(A \cdot \bar{x})$. Observe that given black-box access to f, one can easily simulate black-box access to g, since in order to evaluate g at any input $\alpha \in \mathbb{F}[x_1, x_2, \ldots, x_n]$, one has to simply evaluate f at $A \cdot \alpha$.

Also observe that $g(A^{-1} \cdot \bar{x}) = f(\bar{x})$. Thus any algorithm that can efficiently learn g can also efficiently learn f in the following way. For each $i \in [n]$, suppose that R_i denote the ith row of A^{-1}. Then in the ith input to g simply input the linear polynomial $L_i = \langle R_i, \bar{x} \rangle$, which is the inner product of R_i and the vector $\bar{x}$ of formal input variables. Since g only depends on the first m variables, we only really need to do this operation for $i \in [m]$.

Since f is computed by a degree d multilinear $\Sigma\Pi\Sigma(k)$ circuit, hence $g(\bar{x}) = f(A \cdot \bar{x})$ is also has a natural degree d $\Sigma\Pi\Sigma(k)$ circuit representation, where the linear forms of that representation are obtained by applying the transformation A to corresponding linear forms of C. Let us call this circuit C_g. Notice that C_g may not be multilinear. However, if were somehow able to learn the precise circuit C_g, then by substituting each variable x_i to L_i then we would recover the circuit C_f which is indeed multilinear.

Thus our goal is now the following. We have black-box access to g which only depends on m variables. We would like to devise as algorithm for reconstructing C_g. Now here is a subtle point. C_g is a particular degree d $\Sigma\Pi\Sigma(k)$ representation of g. It has the nice property that when we plug in $x_i = L_i$ in this representation, then we recover a multilinear $\Sigma\Pi\Sigma(k)$ representation of f. Let us call the new object obtained by plugging in $x_i = L_i$ for each i, the "lift" of C_g However, g might have multiple representations as a degree d $\Sigma\Pi\Sigma(k)$ circuit. If given black-box access to g, the reconstruction algorithm finds some other degree d $\Sigma\Pi\Sigma(k)$ representation of g, call it C'_g, then there is no guarantee that when we plug in $x_i = L_i$ in this representation, then we recover a multilinear $\Sigma\Pi\Sigma(k)$ representation of f. In other words, the lift of C'_g may not be multilinear.

Now, we will not actually be able to guarantee that we learn C_g. However the existence of C_g tells us that there exists a $\Sigma\Pi\Sigma(k)$ representation of g whose lift is a multilinear $\Sigma\Pi\Sigma(k)$ circuit. Can we find such a representaion of g?

We will now see that we can actually do this. In order to learn a degree d $\Sigma\Pi\Sigma(k)$ representation of g we will set up a system of polynomial equations whose solution will give as a degree d $\Sigma\Pi\Sigma(k)$ representation. We will be able to impose additional polynomial constraints to this system that will further ensure that whatever $\Sigma\Pi\Sigma(k)$ representation is learnt will be such that its lift will be a multilinear $\Sigma\Pi\Sigma(k)$ circuit.

The algorithm first learns g as a sum of monomials. Since g is of degree at most d and depends on at most kd variable, such a representation of g can be found in time poly $\left(\binom{kd+d}{d}\right)$ using known sparse polynomial reconstruction algorithms [KS01, BOT88]. Let S be the set of m-tuples of non-negative integers that sum to d. Then the algorithm finds a collection of coefficients $\{c_{\bar{e}} \in \mathbb{F} | \bar{e} \in S\}$ such that $g = \sum_{\bar{e} \in S} c_{\bar{e}} \cdot \bar{x}^{\bar{e}}$.

Any degree d $\Sigma\Pi\Sigma(k)$ representation of g looks like the following:

$$\sum_{i=1}^{k} \prod_{j=1}^{d} (a_{j,1}^{(i)} x_1 + a_{j,2}^{(i)} x_2 + \ldots + a_{j,m}^{(i)} x_m + a_{j,m+1}^{(i)}) = \sum_{\bar{e} \in S} c_{\bar{e}} \cdot \bar{x}^{\bar{e}}.$$

The algorithm already knows the set of coefficients $\{c_{\bar{e}} \in \mathbb{F} | \bar{e} \in S\}$. In order to learn a $\Sigma\Pi\Sigma(k)$ representation it needs to learn values for the coefficients in the LHS, i.e. the $a_{j,r}^{(i)}$ for various choices of i, j, r. These $a_{j,r}^{(i)}$ are the unknown variables.

Now for each monomial $x^{\bar{e}}$ that appears in g, we can compare the coefficient of it on the LHS and RHS of the above expression, set them equal to each other and get a polynomial equation in the unknown variables. We do this for all the monomials and hence set up a system of polynomial equations in the unknown variables. Each solution to this system of equations corresponds to a degree d $\Sigma\Pi\Sigma(k)$ representation of g and vice versa.

We are looking for a degree d $\Sigma\Pi\Sigma(k)$ representation whose lift it multilinear. To ensure this, we will add some additional polynomial constraints to our system of polynomial equations.

Now suppose that

$$\sum_{i=1}^{k} \prod_{j=1}^{d} (a_{j,1}^{(i)} x_1 + a_{j,2}^{(i)} x_2 + \ldots + a_{j,m}^{(i)} x_m + a_{j,m+1}^{(i)})$$

represents some degree d $\Sigma\Pi\Sigma(k)$ representation of g. (We still treat the $a_{j,r}^{(i)}$ as unknown variables). In order for its lift to be multilinear, we would need to look at the expression

$$\sum_{i=1}^{k}\prod_{j=1}^{d}(a_{j,1}^{(i)}L_1 + a_{j,2}^{(i)}L_2 + \ldots + a_{j,m}^{(i)}L_m + a_{j,m+1}^{(i)})$$

and in the above expression, any two linear polynomials appearing in the same multiplication gate should be variable disjoint. Now consider a linear polynomial

$$L_j^{(i)} = (a_{j,1}^{(i)}L_1 + a_{j,2}^{(i)}L_2 + \ldots + a_{j,m}^{(i)}L_m + a_{j,m+1}^{(i)})$$

appearing in the expression. Each L_i is a linear form in $x_1, \ldots, x_n$ and the algorithm knows what these L_i are. Thus upon expanding and collecting terms, we see that $L_j^{(i)}$ is a linear polynomial in the $x_1, \ldots, x_n$, with coefficients being linear combinations of $a_{j,1}^{(i)}, a_{j,2}^{(i)}, \ldots, a_{j,m+1}^{(i)}$. Now for the lift to be multilinear, we need that for each $i \in [k]$, $L_1^{(i)}, L_2^{(i)}, \ldots, L_d^{(i)}$ are mutually variable disjoint. In order for $L_j^{(i)}$ and $L_r^{(i)}$ to be variable disjoint, we need to ensure that for each $t \in [n]$, one of the coefficients of x_t in $L_j^{(i)}$ and $L_r^{(i)}$ is zero. Equivalently, it suffices that the product of the coefficient of x_t in $L_j^{(i)}$ and the coefficient of x_t in $L_r^{(i)}$ is zero. This equality is in fact a polynomial constraint in the $a_{j,1}^{(i)}, a_{j,2}^{(i)}, \ldots, a_{j,m+1}^{(i)}$ and the $a_{r,1}^{(i)}, a_{r,2}^{(i)}, \ldots, a_{r,m+1}^{(i)}$ variables.

We add this polynomial equation to our system of polynomial equations. We do this for each $i \in [k]$, for each $j, r \in [d]$ where $j \neq r$, and each $t \in [n]$. Thus we add about $k \cdot d^2 \cdot n$ additional polynomial equations.

Then observe that any solution to the new system will have the property that the lift will be multilinear. Moreover the existence of C_g guarantees that the system will have at least one solution, and hence it solvable in time $\mathrm{Sys}(mdk, kd^2n + \binom{dk+d}{k}, d)$. And the overall time complexity of the algorithm is bounded by $\mathrm{poly}(n, \mathrm{Sys}(d^2k^2, kd^2n + \binom{dk+d}{k}, d)) \leq (dkn)^{\mathcal{O}(d^2k^3)^{(d^2k^2)}}$. □

We observe that Lemma 6.4 can be extended in two aspects: first, as $d \leq \mathrm{rank}(C)$ one can immediately extend the algorithm to the case when the rank ($\mathrm{rank}(C)$) is "small" . It turns out, though, that we can extend the algorithm further to the case when gcd-free-rank ($\Delta_{\mathrm{rank}}(C)$) is small. Note that this is not an immediate extension as one can have a high-degree circuit with constant $\Delta_{\mathrm{rank}}(C)$. One such example would be a circuit in which all the multiplication gates are equal. To avoid such situations we use the algorithm in Corollary 3.10 to factor out $\gcd(C)$ (which is a product of linear functions). Second, we can find a circuit with the smallest possible fan-in, by starting with $k = 1$ and increasing it, until we can found a valid circuit. Note that, we can verify the correctness of our output using Lemma 3.8. The above discussion gives rise the following lemma, the proof of which is left as an easy exercise to the reader.

Lemma 6.5. *Let $f \in \mathbb{F}[x_1, x_2, \ldots, x_n]$ be a polynomial computed by multilinear $\Sigma\Pi\Sigma(k)$ circuit C with $\Delta_{\mathrm{rank}}(C) \leq r$. Then there is a randomized algorithm that given k, r and black-box access to f outputs a multilinear $\Sigma\Pi\Sigma(k')$ circuit computing f, where $k' \leq k$ is the smallest possible fan-in, in time $\mathrm{poly}(n, \mathrm{Sys}(r^2k^2, kr^2n + \binom{rk+r}{k}, r)) \leq (rkn)^{\mathcal{O}(r^2k^3)^{(r^2k^2)}}$.*

Derandomization: The only steps where randomization is required for Lemma 6.4(learning low-degree multilinear $\Sigma\Pi\Sigma(k)$ circuit) are in the variable reduction step (Lemma 3.23) and polynomial system solving (Theorem 3.36). Over $\mathbb{R}$ and $\mathbb{C}$, Theorem 3.36 in fact states that polynomial system solving can be done *deterministically* in the same time complexity.

Moreover, for derandomized variable reduction, we can use Lemma 3.24 instead of Lemma 3.23. Observe that all the assumptions of Lemma 3.24 are satisfied, since a low degree multilinear $\Sigma\Pi\Sigma(k)$ circuit only has constantly many linear forms and hence constantly many essential variables. Moreover the class is closed under taking first order partial derivatives. Furthermore, $\underbrace{\mathcal{C} + \ldots + \mathcal{C}}_{k\,times}$ for $\mathcal{C}$ being the class of multilinear

$Sps(k)$ circuits is just the class if multilinear $\Sigma\Pi\Sigma(k^2)$ circuits, so by Lemma 3.8, there is an efficient hitting set for $\underbrace{\mathcal{C}+\ldots+\mathcal{C}}_{k\,times}$.

Note that, we can also derandomize Lemma 6.5. The only place where we need randomness is used is in the step requiring gcd extraction. Using the deterministic factoring algorithm in [SV10] along with a hitting set of multilinear $\Sigma\Pi\Sigma(k)$ circuits (instead of using the Kaltofen-Trager [KT90] algorithm) gives us a deterministic algorithm for this step.

6.2 Learning High-Degree Multilinear $\Sigma\Pi\Sigma(k)$ Circuits

We show that high-degree case reduces to the low-degree case. More precisely, the high-degree case reduces to the low-gcd-free-rank case, which in turn reduces to the low-degree case. Algorithmically, we invoke Lemma 6.5 together with Lemma 6.19, that simulates a black-box access to all low-gcd-free-rank components of a circuit.

6.2.1 Clustering Algorithm

In [KS09a], a "clustering" algorithm for $\Sigma\Pi\Sigma(k)$ circuits was proposed. Intuitively speaking, this algorithm merges multiplication gates with "high" GCD into clusters. One can also think of these clusters as finding a partition of $[k]$, where all the gates T_1 to T_k which are "close" together according to the Δ_{rank} distance function merge to form a partition of $[k]$. We formalize this notion below:

Definition 6.6 ([KS09a]). *Let C be a multilinear $\Sigma\Pi\Sigma(k)$ circuit and $I = A_1 \cup \ldots \cup A_s = [k]$ be some partition of $[k]$. For each $i \in [s]$, define $C_i \triangleq C_{A_i}$. The set $\{C_i\}_{i=1}^s$ is called* a partition *of C. For $\kappa, r \in \mathbb{N}$, we say a partition $\{C_i\}_{i=1}^s$ is* (κ, r)-strong *when the following conditions hold:*

- $\forall i \in [s],\ \Delta_{\text{rank}}(C_i) \leq r$.
- $\forall i \neq j \in [s],\ \Delta_{\text{rank}}(C_i, C_j) \geq \kappa \cdot r$.

We now give the main relevant result.

Lemma 6.7 (Clustering Algorithm of [KS09a]). *Let $n, k, r_{init}, \kappa \in \mathbb{N}$. There exists an algorithm that given r_{init}, κ and n-variate multilinear $\Sigma\Pi\Sigma(k)$ circuit C as input, outputs $r \in \mathbb{N}$ such that $r_{init} \leq r \leq k^{(k-2)\cdot \log_k(\kappa)} \cdot r_{init}$ and a (κ, r)-strong partition of $[k]$, in time $\mathcal{O}(\log(\kappa) \cdot n^3 k^4)$,*

A key corollary of this result is that for sufficiently (yet, still modestly) large parameters, any two clustered representations of (possible even different) circuits computing the same polynomial are identical (up to a permutation). In that sense, we can say that the clustered representation is unique!

Corollary 6.8 (Implicit in [KS09a]). *Let $n, k, r_{init}, \kappa \in \mathbb{N}$ such that $r_{init} \geq R_M(2k)$ [9] and $\kappa > k^2$, and let C and C' be two minimal multilinear $\Sigma\Pi\Sigma(k)$ circuits computing the same non-zero polynomial. Furthermore, let $C_1, \ldots, C_s$ and $C'_1, \ldots, C'_{s'}$ be the the partitions of C and C', respectively, found by the clustering algorithm on inputs κ, r_{init} together with C and C', respectively. Then $s' = s$ and there exist a permutation $\pi : [s] \to [s]$ such that $\forall i : C_i \equiv C'_{\pi(i)}$.*

Given the above, we can define a *canonical partition* of a circuit.

Definition 6.9. *Let C be a minimal multilinear $\Sigma\Pi\Sigma(k)$ circuit computing a non-zero polynomial. We define $\text{Con}(C) \triangleq (C_1, \ldots C_s)$ as the output of the clustering algorithm, given $r_{init} = R_M(2k), \kappa = k^3$ and C as input.*

[9] $R_M(k)$ is the so-called "Rank Bound" from Theorem 3.12.

Observe that for all $i \in [s] : \Delta_{\text{rank}}(C_i) \leq k^{\mathcal{O}(k)}$. Nonetheless, $\text{Con}(C)$ will never be explicitly computed as it requires the hidden circuit C itself as an input, finding which is the very purpose of the reconstruction algorithm! Yet, a further key observation utilized in [KS09a] is that the uniqueness of clustered representation still holds true if we restrict the circuits to a well-chosen, yet low-dimensional affine space. These are referred to as *rank-preserving subspaces* (a formal definition is given in Definition 6.11). We first state the aforementioned uniqueness property and then discuss rank-preserving subspaces and their constructions in Section 6.2.2.

Lemma 6.10 (Implicit in [KS09a])**.** *Let C be a minimal multilinear $\Sigma\Pi\Sigma(k)$ circuit computing a non-zero polynomial, let $\text{Con}(C) = (C_1, \dots C_s)$ and let V be $k^{\mathcal{O}(k)}$-multilinear-rank-preserving for C. Furthermore, let $C' \equiv C|_V$ and $C'_1, \dots, C'_{s'}$ be the the partition of C' found by the clustering algorithm on inputs $r_{init} = R_M(2k), \kappa = k^3$ and C'. Then $s' = s$ and there exist a permutation $\pi : [s] \to [s]$ such that $\forall i : C'_i = C_{\pi(i)}|_V$.*

6.2.2 Rank Preserving Subspaces

In this section we formalize the notion of multilinear rank-preserving subspaces introduced in [KS08, KS09a] and show new constructions. We begin with a definition.

Definition 6.11 ([KS08, KS09a])**.** *Let $C \equiv \sum_{i=1}^{k} T_i = \sum_{i=1}^{k} \prod_{j=1}^{d_i} \ell_{i,j}$ be a multilinear $\Sigma\Pi\Sigma(k)$ circuit and V an affine subspace. We say that V is* r-multilinear-rank-preserving *for C if the following properties hold:*

1. *For any two linear functions $\ell_{i,j} \nsim \ell_{i',j'}$ appearing in C, we either have that $\ell_{i,j}|_V \nsim \ell_{i',j'}|_V$ or that both $\ell_{i,j}|_V, \ell_{i',j'}|_V$ are constant functions.*
2. *$\forall A \subseteq [k]$, $\text{rank}(\text{sim}(C_A)|_V) \geq \min\{\text{rank}(\text{sim}(C_A)), r\}$.*
3. *No multiplication gate T_i vanishes on V. In other words, for all $i \in [k] : T_i|_V \not\equiv 0$.*
4. *The circuit $C|_V$ is a multilinear circuit.*

In [KS08], a construction of such subspaces was given. Unfortunately, we cannot use this construction directly as it is very "rigid"; we will need something "less structured". Nonetheless, we will build on (and, in fact, generalize) this construction to fit our needs.

Definition 6.12 ([KS08])**.** *For a set $B \subseteq [n]$, we define $V_B \triangleq \text{span}\{e_i \mid i \in B\}$, where $e_i \in \{0,1\}^n$ denotes the i-th standard basis vector.*

This definition was used as the first step of the construction of [KS08]. Indeed, it was shown that it "almost" works.

Lemma 6.13 ([KS08])**.** *Let C be a multilinear $\Sigma\Pi\Sigma(k)$ circuit and $r \in \mathbb{N}$. Then there exists a subset $B \subseteq [n]$ of size $|B| = 2^k \cdot r$ such that for every $B' \supseteq B$ and $\bar{u} \in \mathbb{F}^n$:*

1. *$\forall A \subseteq [k]$, $\text{rank}(\text{sim}(C_A)|_{V_B + \bar{u}}) \geq \min\{\text{rank}(\text{sim}(C_A)), r\}$.*
2. *The circuit $C|_{V_B + \bar{u}}$ is a multilinear circuit.*

The next (and the final) step of the construction of [KS08] was to show that for a particular shift $\bar{u} \in \mathbb{F}^n$, the space $V_B + \bar{u}$ satisfies **all** the requirements of Definition 6.11. In what follows, we generalize this steps by expressing a general condition for $\bar{u} \in \mathbb{F}^n$ under which $V_B + \bar{u}$ satisfies **all** these conditions. Indeed, our result is a direct application of Lemma 3.1. Furthermore, we show a somewhat stronger statement, which in the terminology of [KS08, KS09a] is referred to as "liftable" rank-preserving subspace.

Lemma 6.14. *Let $C \equiv \sum_{i=1}^{k} T_i = \sum_{i=1}^{k} \prod_{j=1}^{d_i} \ell_{i,j}$ be a multilinear $\Sigma\Pi\Sigma(k)$ circuit and let $r \in \mathbb{N}$. Let B be the subset from Lemma 6.13. Then there exist a polynomial $\Phi_C(\bar{x})$ (independent of r and B) of degree less than $2n^3k^2$ such that if $\Phi_C(\bar{u}) \neq 0$ then $V_{B'} + \bar{u}$ is r-multilinear-rank-preserving space for C for every $B' \supseteq B$.*

Proof. Consider the polynomial

$$\Phi_C(\bar{x}) \triangleq \prod_{i=1}^{n} T_i \cdot \prod_{(i,j)\neq(i',j')} D(\ell_{i,j}, \ell_{i',j'}).$$

Here $D(R, L)$ is given by Lemma 3.1. Indeed, Properties 1 and 3 of Definition 6.11 follow from Lemma 3.1. In terms of the degree, observe that there are at most nk linear forms. Therefore,

$$\deg(\Phi_C) \leq nk + \binom{nk}{2} \cdot n < nk + n^3k^2 < 2n^3k^2.$$

□

We conclude this section with two observations. The first observation was implicitly made in [KS08] and was, in fact, used in their construction of rank-preserving subspaces.

Observation 6.15. *For every $C : \Phi_C\left(1, y, y^2, \ldots, y^{n-1}\right)$ is non-zero univariate polynomial in y of degree less than $2n^4k^2$.*

The next observation follows immediately from the definition.

Observation 6.16. *Let $f \in \mathbb{F}[x_1, x_2, \ldots, x_n]$ be a multilinear polynomial, $B \subseteq [n]$ and $\bar{a}, \bar{b} \in \mathbb{F}^n$ two assignments such that $\mathrm{w_H}(\bar{a}, \bar{b}) = 1$. Finally, suppose that $\bar{a}$ and $\bar{b}$ differ (only) in the i-th coordinate. Then:*

1. $\left(f|_{V_{B\cup\{i\}}+\bar{a}}\right)|_{x_i=0} = f|_{V_B+\bar{a}}$
2. $\left(f|_{V_{B\cup\{i\}}+\bar{a}}\right)|_{x_i=b_i-a_i} = f|_{V_B+\bar{b}}$
3. $\left(f|_{V_B+\bar{a}}\right)|_{x_B=\bar{0}_B} = f(\bar{a})$

6.2.3 Cluster Evaluation

For a circuit C, let $\mathrm{Con}(C) = (C_1, \ldots C_s)$ be its canonical partition (see Definition 6.9). Recall that by design each C_i is a "low-rank" circuit. That is, $\Delta_{\mathrm{rank}}(C_i) \leq k^{\mathcal{O}(k)}$. Therefore, if we could evaluate each such C_i on an arbitrary point $\bar{b} \in \mathbb{F}^n$, we could invoke the learning algorithm from Lemma 6.5 and reconstruct it. We show how to achieve this goal via a technique similar to the one used in [BSV20].

We first observe that the uniqueness property w.r.t to rank-preserving spaces (Lemma 6.10) will allow us to evaluate the C_i-s on the space $V_B + \bar{a}$ (and thus on $\bar{a}$) for a "random" point $\bar{a}$ (or a point $\bar{a}$ with a particular structure). Our next step will be to change one (arbitrary) coordinate of such an $\bar{a}$.

Lemma 6.17. *Let C be a multilinear $\Sigma\Pi\Sigma(k)$ circuit and $\mathrm{Con}(C) = (C_1, \ldots C_s)$ be its canonical partition. Then there exists an algorithm that given:*

- *Assignments: $\bar{a}, \bar{b} \in \mathbb{F}^n$ such that $\mathrm{w_H}(\bar{a}, \bar{b}) = 1$ and $\Phi_C(\bar{a}) \neq 0$*
- *The subset $B \subseteq [n]$ of size $k^{\mathcal{O}(k)}$ guaranteed by Lemma 6.13 for $r = k^{\mathcal{O}(k)}$.*
- *Ordered tuple $(C_1|_{V_B+\bar{a}}, \ldots, C_s|_{V_B+\bar{a}})$*

outputs the ordered tuple $\left(C_1|_{V_B+\bar{b}}, \ldots, C_s|_{V_B+\bar{b}}\right)$, in time $n^{k^{k^{\mathcal{O}(k)}}}$.

Proof. Let $i \in [n]$ be coordinate where $\bar{a}$ and $\bar{b}$ differ. The algorithm operates as follows:

- Learn $C' \triangleq C|_{V_{B\cup\{i\}}+\bar{a}}$ using Lemma 6.4 with $d = |B| + 1$.

- Run the Clustering Algorithm from Lemma 6.7 on inputs C', $r_{init} = R_M(2k), \kappa = k^3$. Let $C'_1, \ldots, C'_{s'}$ be the output of the algorithm.
- For $j = 1..s'$: find a coordinate k such that $C'_j|_{x_i=0} = C_k|_{V_B+\bar{a}}$; Set $\sigma(k) \leftarrow j$
- Output $\left(C'_{\sigma(1)}|_{x_i=b_i-a_i}, C'_{\sigma(2)}|_{x_i=b_i-a_i}, \ldots, C'_{\sigma(s')}|_{x_i=b_i-a_i}\right)$

We now argue correctness. By Lemma 6.14, $V_{B\cup\{i\}} + \bar{a}$ is $k^{\mathcal{O}(k)}$-multilinear-rank-preserving for C. Consequently, by Lemma 6.10, $s = s'$ and there exists a permutation $\pi : [s] \to [s]$ such that $\forall j \in [s] : C'_j = C_{\pi(j)}|_{V_{B\cup\{i\}}+\bar{a}}$. By Observation 6.16,

$$\forall j \in [s] : C'_j|_{x_i=0} = C_{\pi(j)}|_{V_B+\bar{a}}.$$

As the clusters in the partition are different, we obtain that $\forall j \in [s] : \pi(j) = k, \sigma(k) = j$ which implies that

$$\forall k \in [s] : \pi(\sigma(k)) = k.$$

Finally, by Observation 6.16:

$$\forall k \in [s] : C'_{\sigma(k)}|_{x_i=b_i-a_i} = \left(C_{\pi(\sigma(k))}|_{V_{B\cup\{i\}}+\bar{a}}\right)|_{x_i=b_i-a_i} = C_{\pi(\sigma(k))}|_{V_B+\bar{b}} = C_k|_{V_B+\bar{b}}.$$

The running time follows from Lemmas 6.4 and 6.7. □

Next, as in [BSV20], by applying the lemma iteratively we can extend the evaluation algorithm to handle assignments with *arbitrary* Hamming distance, yet under some technical conditions. This can be considered as a grass-hopper jump. To formulate these conditions, we will use the notations from Definition 3.2.

Corollary 6.18. *Let C be a multilinear $\Sigma\Pi\Sigma(k)$ circuit and $\mathrm{Con}(C) = (C_1, \ldots C_s)$ be its canonical partition. Then there exists an algorithm that given:*

- *Assignments: $\bar{a}, \bar{b} \in \mathbb{F}^n$ such that for all $0 \le i \le n-1$, $\Phi_C(\gamma^i(\bar{a}, \bar{b})) \neq 0$.*
- *The subset $B \subseteq [n]$ of size $k^{\mathcal{O}(k)}$ guaranteed by Lemma 6.13 for $r = k^{\mathcal{O}(k)}$.*
- *Ordered tuple $(C_1|_{V_B+\bar{a}}, \ldots, C_s|_{V_B+\bar{a}})$*

outputs the ordered tuple $\left(C_1|_{V_B+\bar{b}}, \ldots, C_s|_{V_B+\bar{b}}\right)$ and hence $\left(C_1(\bar{b}), \ldots, C_s(\bar{b})\right) \in \mathbb{F}^s$, in time $n^{k^{k^{\mathcal{O}(k)}}}$.

Proof. Apply Lemma 6.17 iteratively, using the ordered tuple $\left(C_1|_{V_B+\gamma^i(\bar{a},\bar{b})}, \ldots, C_s|_{V_B+\gamma^i(\bar{a},\bar{b})}\right)$ to compute the ordered tuple $\left(C_1|_{V_B+\gamma^{i+1}(\bar{a},\bar{b})}, \ldots, C_s|_{V_B+\gamma^{i+1}(\bar{a},\bar{b})}\right)$, for $0 \le i \le n-1$, recalling that $\bar{a} = \gamma^0(\bar{a}, \bar{b})$ and $\bar{b} = \gamma^n(\bar{a}, \bar{b})$. The last part follows from Observation 6.16. □

Now we show how to evaluate $\mathrm{Con}(C)$ on $V|_{B+\bar{b}}$ for an *arbitrary* $\bar{b} \in \mathbb{F}^n$ and hence $\mathrm{Con}(C)|_{\bar{x}=\bar{b}}$. In order to do this, we will consider the line $\ell_{\bar{a},\bar{b}}(t)$ through $\bar{a}$ and $\bar{b}$ and show that "most" points $\bar{u}$ on this line do satisfy the condition that $\Phi_C(\bar{u}) \neq 0$. Once we have this, by Corollary 6.18, we will show that for most points $\bar{u}$ on the line, $\mathrm{Con}(C)|_{B+\bar{u}}$ can be computed accurately. We then apply noisy polynomial interpolation (for instance the Berlekamp-Welch algorithm for decoding Reed-Solomon Codes) to recover the entire univariate polynomial which is $\mathrm{Con}(C)$ restricted to $V|_{B+\ell_{\bar{a},\bar{b}}(t)}$, and from this we can recover $\mathrm{Con}(C)$ on $V|_{B+\bar{b}}$, and hence on $\bar{b}$.

Lemma 6.19. *Let C be a multilinear $\Sigma\Pi\Sigma(k)$ circuit and $\mathrm{Con}(C) = (C_1, \ldots C_s)$ be its canonical partition. Then there exists an algorithm that given:*

- *Assignments: $\bar{a}, \bar{b} \in \mathbb{F}^n$ such that $\Phi_C(\bar{a}) \neq 0$.*

- *The subset $B \subseteq [n]$ of size $k^{\mathcal{O}(k)}$ guaranteed by Lemma 6.13 for $r = k^{\mathcal{O}(k)}$.*
- *Ordered tuple $(C_1|_{V_B+\bar{a}}, \ldots, C_s|_{V_B+\bar{a}})$*

outputs the ordered tuple $\left(C_1(\bar{b}), \ldots, C_s(\bar{b})\right) \in \mathbb{F}^s$ in time $n^{k^{k^{\mathcal{O}(k)}}}$.

The algorithm and the proof mimic Lemma 5.4 from [BSV20].

Proof. Let $W \subseteq \mathbb{F}$ be a subset of size $|W| = 5n^4k^2$ and let $f = (f_1, \ldots, f_s) : \mathbb{F} \to \mathbb{F}^s$ be a function to be specified later. The algorithm operates as follows:

- For each $\alpha \in W$, define $f(\alpha)$ as the output of the algorithm in Corollary 6.18 for the input assignments $\bar{a}$ and $\bar{u} = \ell_{\bar{a},\bar{b}}(\alpha)$.
- For $i \in [s]$: use noisy polynomial interpolation (Lemma 3.3) on f_i to recover a polynomial $\hat{f}_i(t)$ of degree at most n
- Output $\left(\hat{f}_1(1), \ldots, \hat{f}_s(1)\right)$

We now analyse the algorithm. Consider the following polynomials:

$$Q(t) \triangleq \prod_{i=1}^{n-1} \Phi_C\left(\gamma^i(\bar{a}, \ell_{\bar{a},\bar{b}}(t))\right) \ , \ P_i(t) \triangleq C_i(\ell_{\bar{a},\bar{b}}(t)) \text{ for } i \in [s].$$

Observe that by Corollary 6.18, if $Q(\alpha) \neq 0$ then $\forall i \in [s] : f_i(\alpha) = P_i(\alpha)$. We will now bound the number of roots of $Q(t)$. By Lemma 6.14, $Q(t)$ is a univariate polynomial of degree less that $2n^4k^2$. In addition, $Q \not\equiv 0$ since $Q(0) = (\Phi_C(\bar{a}))^n \neq 0$. Consequently, $Q(t)$ has less that $2n^4k^2$ roots. On the other hand, for every $i \in [s] : P_i(t)$ is a univariate polynomial of degree at most n. By Lemma 3.3, for each $i \in [s] : \hat{f}_i(t) \equiv P_i(t)$. In particular, $\hat{f}_i(1) = P_i(1) = C_i(\bar{b})$. □

6.2.4 Putting all together

We can finally prove Theorem 1.6.

Proof. (of Theorem 1.6.) Let $W \subseteq \mathbb{F}$ be a subset of size $|W| = 2n^4k^2$. The algorithm operates as follows:

Repeat the following steps for each $\alpha \in W$ and a subset $B \subseteq [n]$ of size $|B| = k^{\mathcal{O}(k)}$:

- Let $\bar{a} \triangleq (1, \alpha, \alpha^2, \ldots, \alpha^{n-1})$
- Learn $C' \triangleq C|_{V_B+\bar{a}}$ using Lemma 6.4 with $d = |B|$
- Run the Clustering algorithm from Lemma 6.7 on inputs C', $r_{init} = R_M(2k), \kappa = k^3$.
  Let $C'_1, \ldots, C'_{s'}$ be the output of the algorithm.
- For each $i \in [s']$ use the algorithm from Lemma 6.5 on C_i with $r = k^{\mathcal{O}(k)}$ to output a circuit $\hat{C}_i$.
  Use Lemma 6.19 with $\bar{a}, B$ and $(C'_1, \ldots, C'_{s'})$, as inputs to simulate black-box access to C_i
- Let $\hat{C} \triangleq \hat{C}_1 + \ldots + \hat{C}_{s'}$
- If $C \equiv \hat{C}$ (using Lemma 3.8) and $\hat{C}$ has top fan-in $\leq k$, output $\hat{C}$;
  otherwise, continue to the next iteration.

We now analyze the algorithm. First, observe that the algorithm can only output a multilinear $\Sigma\Pi\Sigma(k)$ circuit that is equivalent to C. We will now argue that there exist at least one iteration when such a circuit is computed.

Let B be the set guaranteed by Lemma 6.13 for $r = k^{\mathcal{O}(k)}$. Furthermore, by Observation 6.15, there exists $\alpha \in W$ such that $\Phi_C(\bar{a}) \neq 0$ for $\bar{a} = (1, \alpha, \alpha^2, \ldots, \alpha^{n-1})$. By Lemma 6.14, $V_B + \bar{a}$ is $k^{\mathcal{O}(k)}$-multilinear-rank-preserving for C. Thus by Lemma 6.10, $s = s'$ and there exists a permutation $\pi : [s] \to [s]$ such that $\forall i \in [s] : C'_i = C_{\pi(i)}|_{V_B + \bar{a}}$. Assume WLOG that $\forall i : \pi(i) = i$ [10]. Given that, Lemma 6.19 guarantees black-box access to $\mathrm{Con}(C)$. Recall that $\forall i \in [s] : \Delta_{\mathrm{rank}}(C_i) \leq k^{\mathcal{O}(k)}$. Consequently, by Lemma 6.5 $\forall i \in [s] : \hat{C}_i \equiv C_i$ and hence $\hat{C} = \hat{C}_1 + \ldots + \hat{C}_s \equiv C_1 + \ldots C_s = C$. Finally, by the minimality property of Lemma 6.5, for each $i \in [s]$ the fan-in of $\hat{C}_i$ is at most the fan-in of C_i. hence, the fan-in of $\hat{C}$ is at most k. Consequently, for the above choices of α and B the algorithm will output a circuit, as required.

□

Derandomization: The only place in the entire algorithm where randomness was used was in Lemmas 6.4 and 6.5. At the end of those lemmas we already commented on how they can be derandomized over $\mathbb{R}$ and $\mathbb{C}$.

References

[Agr05] M. Agrawal. Proving lower bounds via pseudo-random generators. In *Proceedings of the 25th FSTTCS*, volume 3821 of *LNCS*, pages 92–105, 2005.

[AKV20] N. Alon, M. Kumar, and B. L. Volk. Unbalancing sets and an almost quadratic lower bound for syntactically multilinear arithmetic circuits. *Comb.*, 40(2):149–178, 2020.

[AM94] L. M. Adleman and K. S. McCurley. Open problems in number theoretic complexity, ii. In *International Algorithmic Number Theory Symposium*, pages 291–322. Springer, 1994.

[AM10] V. Arvind and P. Mukhopadhyay. The monomial ideal membership problem and polynomial identity testing. *Information and Computation*, 208(4):351–363, 2010.

[Ang88] D. Angluin. Queries and concept learning. *Machine Learning*, 2:319–342, 1988.

[AV08] M. Agrawal and V. Vinay. Arithmetic circuits: A chasm at depth four. In *Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 67–75, 2008.

[AvMV15] M. Anderson, D. van Melkebeek, and I. Volkovich. Derandomizing polynomial identity testing for multilinear constant-read formulae. *Computational Complexity*, 24(4):695–776, 2015.

[BBB+00] A. Beimel, F. Bergadano, N. H. Bshouty, E. Kushilevitz, and S. Varricchio. Learning functions represented as multiplicity automata. *J. ACM*, 47(3):506–530, 2000.

[BOT88] M. Ben-Or and P. Tiwari. A deterministic algorithm for sparse multivariate polynominal interpolation. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing (STOC)*, pages 301–309, 1988.

[BSV20] V. Bhargava, S. Saraf, and I. Volkovich. Reconstruction of depth-4 multilinear circuits. In Shuchi Chawla, editor, *Proceedings of the 31st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2144–2160. SIAM, 2020.

[Can88] J. Canny. Some algebraic and geometric computations in pspace. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 460–467, 1988.

[10] One could define $\mathrm{Con}(C)$ up to a permutation. All the previous analyses would carry over for a fixed permutation.

[Car06] E. Carlini. Reducing the number of variables of a polynomial. In Mohamed Elkadi, Bernard Mourrain, and Ragni Piene, editors, *Algebraic Geometry and Geometric Modeling*, pages 237–247. Springer, 2006.

[CIKK16] M. L. Carmosino, R. Impagliazzo, V. Kabanets, and A. Kolokolova. Learning algorithms from natural proofs. In *Proceedings of the 31st Conference on Computational Complexity, CCC*, pages 1–24, 2016.

[CLO15] D. A. Cox, J. Little, and D. O'Shea. *Ideals, varieties, and algorithms - an introduction to computational algebraic geometry and commutative algebra (4. ed.)*. Undergraduate texts in mathematics. Springer, 2015.

[CM20] S. Chen and R. Meka. Learning polynomials in few relevant dimensions. In Jacob D. Abernethy and Shivani Agarwal, editors, *Conference on Learning Theory, COLT 2020, 9-12 July 2020, Virtual Event [Graz, Austria]*, volume 125 of *Proceedings of Machine Learning Research*, pages 1161–1227. PMLR, 2020.

[DL78] R. A. DeMillo and R. J. Lipton. A probabilistic remark on algebraic program testing. *Inf. Process. Lett.*, 7(4):193–195, 1978.

[DS07] Z. Dvir and A. Shpilka. Locally decodable codes with 2 queries and polynomial identity testing for depth 3 circuits. *SIAM J. on Computing*, 36(5):1404–1434, 2007.

[FK09] L. Fortnow and A. R. Klivans. Efficient learning algorithms yield circuit lower bounds. *J. Comput. Syst. Sci.*, 75(1):27–36, 2009.

[FS12] M. A. Forbes and A. Shpilka. Quasipolynomial-time identity testing of non-commutative and read-once oblivious algebraic branching programs. *Electronic Colloquium on Computational Complexity (ECCC)*, 19:115, 2012.

[FSS14] M. A. Forbes, R. Saptharishi, and A. Shpilka. Hitting sets for multilinear read-once algebraic branching programs, in any order. In *Symposium on Theory of Computing, STOC*, pages 867–875, 2014.

[GKKS13] A. Gupta, P. Kamath, N. Kayal, and R. Saptharishi. Arithmetic circuits: A chasm at depth three. In *Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 578–587, 2013.

[GKL11] A. Gupta, N. Kayal, and S. V. Lokam. Efficient reconstruction of random multilinear formulas. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS*, pages 778–787, 2011.

[GKL12] A. Gupta, N. Kayal, and S. V. Lokam. Reconstruction of depth-4 multilinear circuits with top fanin 2. In *Proceedings of the 44th Annual ACM Symposium on Theory of Computing (STOC)*, pages 625–642, 2012. Full version at https://eccc.weizmann.ac.il/report/2011/153.

[GKQ14] A. Gupta, N. Kayal, and Y. Qiao. Random arithmetic formulas can be reconstructed efficiently. *Computational Complexity*, 23(2):207–303, 2014.

[GKS20] A. Garg, N. Kayal, and C. Saha. Learning sums of powers of low-degree polynomials in the non-degenerate case. *arXiv preprint arXiv:2004.06898*, 2020.

[GV88] D. Yu. Grigor'ev and N.N. Vorobjov. Solving systems of polynomial inequalities in subexponential time. *Journal of Symbolic Computation*, 5(1):37 – 64, 1988.

[Hås90] Johan Håstad. Tensor rank is np-complete. *J. Algorithms*, 11(4):644–654, 1990.

[HW99] M. D. Huang and Y. C. Wong. Solvability of systems of polynomial congruences modulo a large prime. *computational complexity*, 8(3):227–257, 1999.

[Ier89] D. Ierardi. Quantifier elimination in the theory of an algebraically-closed field. In *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing*, STOC '89, page 138–147, New York, NY, USA, 1989. Association for Computing Machinery.

[Kap57] I. Kaplansky. *An Introduction to Differential Algebra*. Hermann, Paris, 1957.

[Kay11] N. Kayal. Efficient algorithms for some special cases of the polynomial equivalence problem. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete algorithms*, pages 1409–1421. SIAM, 2011.

[KI03] V. Kabanets and R. Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC)*, pages 355–364, 2003.

[KNS18] N. Kayal, V. Nair, and C. Saha. Average-case linear matrix factorization and reconstruction of low width algebraic branching programs. *Electronic Colloquium on Computational Complexity (ECCC)*, 25:29, 2018.

[KNST17] N. Kayal, V. Nair, C. Saha, and S. Tavenas. Reconstruction of full rank algebraic branching programs. In *32nd Computational Complexity Conference, CCC 2017.*, pages 21:1–21:61, 2017.

[Koi96] P. Koiran. Hilbert's nullstellensatz is in the polynomial hierarchy. *Journal of complexity*, 12(4):273–286, 1996.

[Koi10] P. Koiran. Arithmetic circuits: the chasm at depth four gets wider. *CoRR*, abs/1006.4700, 2010.

[KS01] A. Klivans and D. Spielman. Randomness efficient identity testing of multivariate polynomials. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 216–223, 2001.

[KS06] A. Klivans and A. Shpilka. Learning restricted models of arithmetic circuits. *Theory of computing*, 2(10):185–206, 2006.

[KS07] N. Kayal and N. Saxena. Polynomial identity testing for depth 3 circuits. *Computational Complexity*, 16(2):115–138, 2007.

[KS08] Z. S. Karnin and A. Shpilka. Deterministic black box polynomial identity testing of depth-3 arithmetic circuits with bounded top fan-in. In *Proceedings of the 23rd Annual IEEE Conference on Computational Complexity (CCC)*, pages 280–291, 2008.

[KS09a] Z. S. Karnin and A. Shpilka. Reconstruction of generalized depth-3 arithmetic circuits with bounded top fan-in. In *Proceedings of the 24th Annual IEEE Conference on Computational Complexity (CCC)*, pages 274–285, 2009. Full version at http://www.cs.technion.ac.il/ shpilka/publications/KarninShpilka09.pdf.

[KS09b] N. Kayal and S. Saraf. Blackbox polynomial identity testing for depth 3 circuits. In *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 198–207, 2009. Full version at https://eccc.weizmann.ac.il/report/2009/032.

[KS09c] A. R. Klivans and A. A. Sherstov. Cryptographic hardness for learning intersections of halfspaces. *J. Comput. Syst. Sci.*, 75(1):2–12, 2009.

[KS11] Z. S. Karnin and A. Shpilka. Black box polynomial identity testing of generalized depth-3 arithmetic circuits with bounded top fan-in. *Combinatorica*, 31(3):333–364, 2011.

[KS19] N. Kayal and C. Saha. Reconstruction of non-degenerate homogeneous depth three circuits. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019.*, pages 413–424, 2019.

[KT90] E. Kaltofen and B. M. Trager. Computing with polynomials given by black boxes for their evaluations: Greatest common divisors, factorization, separation of numerators and denominators. *J. of Symbolic Computation*, 9(3):301–320, 1990.

[Lan12] J. Landsberg. Tensors: geometry and applications. *Representation theory*, 381(402):3, 2012.

[MR75] Y. Matijasevič and J. Robinson. Reduction of an arbitrary diophantine equation to one in 13 unknowns. *Acta Arithmetica*, 27(1):521–553, 1975.

[Raz13] R. Raz. Tensor-rank and lower bounds for arithmetic formulas. *J. ACM*, 60(6):40:1–40:15, 2013.

[Sch80] J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, 1980.

[Shi16] Y. Shitov. How hard is the tensor rank? *arXiv preprint arXiv:1611.01559*, 2016.

[Shp09] A. Shpilka. Interpolation of depth-3 arithmetic circuits with two multiplication gates. *SIAM J. on Computing*, 38(6):2130–2161, 2009.

[Sin16] G. Sinha. Reconstruction of real depth-3 circuits with top fan-in 2. In *31st Conference on Computational Complexity, CCC 2016, May 29 to June 1, 2016, Tokyo, Japan*, pages 31:1–31:53, 2016.

[Sin20] G. Sinha. Efficient reconstruction of depth three circuits with top fan-in two. *Electron. Colloquium Comput. Complex.*, 27:125, 2020.

[SS09] N. Saxena and C. Seshadhri. An almost optimal rank bound for depth-3 identities. In *Proceedings of the 24th Annual IEEE Conference on Computational Complexity (CCC)*, pages 137–148, 2009.

[SS10] N. Saxena and C. Seshadhri. From Sylvester-Gallai Configurations to Rank Bounds: Improved Black-Box Identity Test for Deph-3 Circuits. In *Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 21–30, 2010.

[SS11] N. Saxena and C. Seshadhri. An almost optimal rank bound for depth-3 identities. *SIAM J. Comput.*, 40(1):200–224, 2011.

[SS12] N. Saxena and C. Seshadhri. Blackbox identity testing for bounded top-fanin depth-3 circuits: The field doesn't matter. *SIAM J. Comput.*, 41(5):1285–1298, 2012.

[SS13] N. Saxena and C. Seshadhri. From sylvester-gallai configurations to rank bounds: Improved blackbox identity test for depth-3 circuits. *J. ACM*, 60(5):33, 2013.

[SS16] M. Schaefer and D. Stefankovic. The complexity of tensor rank. *CoRR*, abs/1612.04338, 2016.

[Sud98] M. Sudan. Algebra and computation. http://people.csail.mit.edu/madhu/FT98/course.html, 1998. Lecture notes.

[SV10] A. Shpilka and I. Volkovich. On the relation between polynomial identity testing and finding variable disjoint factors. In *Automata, Languages and Programming, 37th International Colloquium (ICALP)*, pages 408–419, 2010. Full version at https://eccc.weizmann.ac.il/report/2010/036.

[SV15] A. Shpilka and I. Volkovich. Read-once polynomial identity testing. *Computational Complexity*, 24(3):477–532, 2015.

[SV18] S. Saraf and I. Volkovich. Blackbox identity testing for depth-4 multilinear circuits. *Combinatorica*, 38(5):1205–1238, 2018.

[SY10] A. Shpilka and A. Yehudayoff. Arithmetic circuits: A survey of recent results and open questions. *Foundations and Trends in Theoretical Computer Science*, 5(3-4):207–388, 2010.

[Tav13] S. Tavenas. Improved bounds for reduction to depth 4 and depth 3. In *MFCS*, pages 813–824, 2013.

[Zip79] R. Zippel. Probabilistic algorithms for sparse polynomials. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation*, pages 216–226, 1979.

A Solving system of algebraic equations using elimination theory

The aim of this section is to show that given a system of m polynomial equations $\{f_1 = 0,\ f_2 = 0, \ldots,\ f_m = 0\}$ where $f_i \in \mathbb{F}[x_1, x_2, \ldots, x_n]$ of degree d, there exists a randomized $\text{poly}((dnm)^{3^n})$-time that outputs a point $\bar{a} \in \bar{\mathbb{F}}$ if it exists, and otherwise outputs "no solution". Also, the degree of extension of the solution(outputted our algorithm) is bounded by $\text{poly}((dn)^{3^n})$.

The algorithm we present in this section is based on elimination theory and extension theorem which are well known in algebraic geometry literature [CLO15]. This algorithm is recursive in nature: essentially we reduce a system on n variate polynomial to $n-1$ variate polynomial system and so on. When we reach $n = 1$, we can use the fact that univariate systems are easy to solve[11], thus concluding our algorithm.

The algorithm we describe below has some corner cases, essentially to ensure that we get a non-trivial resultant. We will elaborate on each of them below:

1. $m = 1$: Then we can't take resultant as it requires atleast 2 polynomials. However, we can just substitute n-1 variable to random values and find the solution. Note that, the solution we found here will be over an extension of degree atmost d.

2. $\gcd(f_1, f_2, \ldots, f_m) \neq 1$: If $\gcd(f_1, f_2, \ldots, f_m) \neq 1$ then resultant as needed in our algorithm will turn out to be identically 0, which gives a trivial System 10. To overcome this, strip off any common gcd and then solve two separate systems, $\{f_1' = 0,\ f_2' = 0, \ldots,\ f_m' = 0\}$ and $\{gcd(f_1, f_2, \ldots, f_m) = 0\}$, where $f_i' = \frac{f_i}{gcd(f_1, f_2, \ldots, f_m)}$. Note that, a solution to either system will give us a solution to $\{f_1 = 0,\ f_2 = 0, \ldots,\ f_m = 0\}$.

3. $m > \binom{n+d}{d}$: Note that, the dimension of the space of n variate degree d polynomials is $\binom{n+d}{d}$. Thus, we can always ensure that $m \leq \binom{n+d}{d}$ by removing any redundant/dependent f_i, in $\text{poly}\binom{n+d}{d}$ time.

Now that we have discussed all the corner cases, we will assume that $\binom{n+d}{d} \geq m > 1$ and $\gcd(f_1, f_2, \ldots, f_m) = 1$, we can proceed to discussing the core idea of this approach. That is, when we convert our n-variate system to $n-1$ variate such that it preserves the solutions. And, that each solution to $n-1$ variate system is extendable. This is exactly what we show in next lemma.

The lemma also assumes that f_i-s are monic in x_1. Note that, this can be ensured by the following shift in variables $x_i = x_i + a_i x_1$ for random a_i-s.

$$\forall i \in [m], \qquad f_i(x) = 0, \tag{9}$$

Define $h \in \mathbb{F}[\bar{x}, \bar{u}] := Res_{x_1}(f_1, u_2 f_2 + \ldots + u_m f_m) = \sum_\alpha h_\alpha u^\alpha$.

Since we have already taken care of the case when $m = 1$ (1), we can assume that $m > 1$ thus ensuring that h is well defined. Also, $h \neq 0$ because of corner-case 2.

$$\forall \alpha \qquad h_\alpha(x) = 0 \tag{10}$$

[11] Solving a univariate system is equivalent to factoring the univariate polynomials and finding a non-trivial gcd.

Lemma A.1. *System 9 has solutions iff System 10 has solutions.*

Proof. System 9 $\implies$ System 10: Let $(a, \bar{c})$ be a solution to 9, since all f_i's are monic, we get that there exist a non-trivial gcd among $f_1(x_1, \bar{c}), f_2(x_1, \bar{c}), \ldots, f_m(x_m, \bar{c})$. This in-turn implies $Res_{x_1}(f_1(x_1, \bar{c}), u_2 f_2(x_1, \bar{c}) + u_3 f_3(x_1, \bar{c}) + \ldots u_m f_m(x_1, \bar{c})) \equiv 0$. Thus system 10 will have solutions. Note that if $(a, \bar{c}) \in \mathbb{G}^n$ then system 10 also has a solution over $\mathbb{G}$, where $\mathbb{G}$ is some extension of $\mathbb{F}$.

System 10 $\implies$ System 9: Let $\bar{c'} \in \mathbb{F}\, s.t.\, h_\alpha(\bar{c'}) = 0\, \forall \bar{\alpha}$. Note that, due to monicness assumption, $h(\bar{c'}, u_2, \ldots u_m) = Res_{x_1}(f_1(x_1, \bar{c'}), u_2 f_2(x_1, \bar{c'}) + u_3 f_3(x_1, \bar{c'}) + \ldots u_m f_m(x_1, \bar{c'}))$.

Since, $h_\alpha(\bar{c'}) = 0\, \forall \bar{\alpha} = 0$, we get that,

$$Res_{x_1}(f_1(x_1, \bar{c'}), u_2 f_2(x_1, \bar{c'}) + u_3 f_3(x_1, \bar{c'}) + \ldots u_m f_m(x_1, \bar{c'})) = 0.$$

This implies, there exist a common factor $\mathcal{F}$ with positive degree in x_1 of $f_1(x_1, \bar{c'})$ and $u_2 f_2(x_1, \bar{c'}) + u_3 f_3(x_1, \bar{c'}) + \ldots u_m f_m(x_1, \bar{c'})$.

Since $\mathcal{F} | f_1(x_1, \bar{c'})$ we get that $\mathcal{F} \in \mathbb{F}[x_1]$. By comparing coefficients of u_i-s on both sides in the following equation,

$$\mathcal{F}(x_1) A(x_1, u_2, \ldots, u_m) = u_2 f_2(x_1, \bar{c'}) + u_3 f_3(x_1, \bar{c'}) + \ldots u_m f_m(x_1, \bar{c'}),$$

thus we get that $\mathcal{F} | f_i(x_1, \bar{c'})$, for $i > 1$. As a direct consequence, we that $(\zeta, \bar{c'})$ is a solution of system 9, where ζ is a root of $\mathcal{F}$. Note that if $\bar{c'} \in \mathbb{G}^{n-1}$ then ζ lies in just in a degree $2d^2$ extension of $\mathbb{G}$, where $\mathbb{G}$ is some extension of $\mathbb{F}$. □

We will now write the skeleton for a recursive algorithm to solve system 9. Its correctness follows from what we have discussed above.

Algorithm 1: $Algorithm_{\mathbb{F}}(\{f_1, f_2, \ldots f_m\}, n, d)$

Input: $f_1, f_2, \ldots f_m \in \mathbb{F}[x_1, x_2, \ldots, x_n]$ s.t each f_i is monic in x_1 and total degree atmost d.
Output: *Simultaneous solution to* $f_i(\bar{x}) = 0$
  if $m = 1$: **then**
    See corner-case 1.
  else
    Check for non-trivial gcd using factoring. See corner-case 2.
    Compute $Res_{x_1}(f_1, u_2 f_2 + \ldots + u_m f_m) = \sum_\alpha h_\alpha u^\alpha$
    Find $\bar{c}'$ s.t. $\forall \alpha\, h_\alpha(\bar{c}') = 0$ using $ALG(\{h_\alpha\}, n-1, 2d^2)$.
    If $ALG(\{h_\alpha\}, n-1, 2d^2)$ *fails* then output no solution, else solve for x_1 in
    $f_1(x_1, \bar{c}') = 0, \ldots, f_i(x_1, \bar{c}') = 0$. Let the output be ζ.
    return $(\zeta, \bar{c}')$.
  end if

Time complexity: Note that, we have dropped m from the list of parameters as $m \leq \binom{n+d}{d}$. Note that, the time complexity $T(n, d)$ of Algorithm 1 satisfies the following inequality. $T(n, d) \leq poly(\binom{n+d}{d}) + T(n - 1, 2d^2)$.

Thus, $T(n, d) \leq \text{poly}((dnm)^{3^n})$. Similar analysis also gives that the degree of extension of the solution(outputted by algorithm 1) is bounded by $\text{poly}((dn)^{3^n})$.

ECCC ISSN 1433-8092
https://eccc.weizmann.ac.il