

Interpolation of Depth-3 Arithmetic Circuits with Two Multiplication Gates*

Amir Shpilka[†]

Abstract

In this paper we consider the problem of constructing a small arithmetic circuit for a polynomial for which we have oracle access. Our focus is on n -variate polynomials, over a finite field \mathbb{F} , that have depth-3 arithmetic circuits (with an addition gate at the top) with two multiplication gates of degree at most d . We obtain the following results:

1. Multilinear case: When the circuit is multilinear (multiplication gates compute multilinear polynomials) we give an algorithm that outputs, with probability $1 - o(1)$, *all* the depth-3 circuits with two multiplication gates computing the polynomial. The running time of the algorithm is $\text{poly}(n, |\mathbb{F}|)$.
2. General case: When the circuit is not multilinear we give a quasi-polynomial (in $n, d, |\mathbb{F}|$) time algorithm that outputs, with probability $1 - o(1)$, a succinct representation of the polynomial. In particular, if the depth-3 circuit for the polynomial is not of small *depth-3 rank* (namely, after removing the g.c.d. of the two multiplication gates, the remaining linear functions span a not too small linear space) then we output the depth-3 circuit itself. In case that the rank is small we output a depth-3 circuit with a quasi-polynomial number of multiplication gates.

Prior to our work there have been several interpolation algorithms for restricted models. However, all the techniques used there completely fail when dealing with depth-3 circuits with even just two multiplication gates. Our proof technique is new and relies on the factorization algorithm for multivariate black-box polynomials, on lower bounds on the length of linear locally decodable codes with 2 queries, and on a theorem regarding the structure of identically zero depth-3 circuits with four multiplication gates.

*Preliminary version appeared in [Shp07].

[†]Faculty of Computer Science, Technion, Haifa 32000, Israel. Email: shpilka@cs.technion.ac.il. This research was supported by the Israel Science Foundation (grant number 439/06).

1 Introduction

In this work we consider the problem of constructing a small arithmetic circuit for a polynomial, for which we have oracle access. That is, there is a black box holding a polynomial f and we would like to find a small arithmetic circuit that computes f . We are allowed to pick inputs (adaptively) and query the black box for the value of f on those inputs. The focus of this work is on n -variate polynomials that have small depth-3 arithmetic circuits¹, over some finite field \mathbb{F} . We consider the *simplest* such circuits, those with only two multiplication gates (also known as $\Sigma\Pi\Sigma(2)$ circuits). We obtain the following results. Let f be a polynomial, for which we have oracle access, that is computed by a $\Sigma\Pi\Sigma(2)$ circuit. When f is computed by a *multilinear* $\Sigma\Pi\Sigma(2)$ circuit we give a polynomial time algorithm that outputs, with high probability, *all* the multilinear $\Sigma\Pi\Sigma(2)$ circuits that compute f . When f does not have a multilinear $\Sigma\Pi\Sigma(2)$ circuit, we output in quasi-polynomial time, w.h.p., a short description for f (depending on a technical condition we either output a $\Sigma\Pi\Sigma(2)$ circuit for it, or a depth-3 circuit of quasi-polynomial size).

The problem of reconstructing a small arithmetic circuit for a polynomial using queries is a basic problem in algebraic complexity and is closely related to problems in learning theory. We now give some background that explains why studying depth-3 circuits is the next natural step given our current state of knowledge.

1.1 Computational learning theory

This paper considers the task of exact learning of algebraic functions. The analogous problem for Boolean functions is well-studied and we first summarize the state of knowledge there. The question of whether we can compute a small description for a boolean function, for which we have oracle access, is a fundamental problem in learning theory. The problem, also known as the exact learning problem using membership queries, attracted a lot of research and both positive and negative results were proved. On the negative side it was shown that if a class of boolean circuits \mathcal{C} contains *trapdoor functions* or *pseudo-random functions* then there are no efficient learning algorithms for it [OGM86, KV94, Kha95]. In particular, there are no efficient interpolation algorithms for the class TC_4^0 (the class of depth 4 threshold circuits), under a widely believed cryptographic assumption [KL01, NR04]. Moreover, in [RR97] it was proved that if we consider a class of circuits \mathcal{C} that can compute pseudo-random functions efficiently then it is hard to determine, in exponential time, whether a function given by its truth table can be computed efficiently by a circuit from \mathcal{C} . In other words, even if the algorithm is given the whole truth table as input, it cannot determine whether f has a polynomial size circuit in \mathcal{C} or not, in exponential time (i.e., in time polynomial in the size of the truth table).

¹In this work whenever we say depth-3 circuit we mean a circuit with an addition gate at the top. The reason being that by factoring the circuit and then applying known interpolation algorithms for depth-2 circuits it is easy to reconstruct depth-3 circuits that have a multiplication gate at the top.

On the positive side, there are many works showing that in some restricted models of computation, e.g. when f has a small circuit from a restricted class of circuits, exact learning from membership queries is possible (e.g. [SS96, BBV96, BBTV97, BBB⁺00]). However, no exact learning algorithms are known for the class of bounded depth boolean circuits. Moreover, even if we allow the algorithm to run in exponential time and have access to the truth table of the function then it is still not known how to compute a small bounded depth circuit for it.

To conclude, exact learning is known only for very restricted classes of circuits, and we cannot hope to learn the class of depth-4 threshold circuits if certain cryptographic assumptions hold.

1.2 Interpolation of arithmetic circuits

As mentioned above we consider the algebraic analog of the exact learning problem. Let \mathcal{A} be a class of arithmetic circuits over a field \mathbb{F} . We are given oracle access to a polynomial $f(x_1, \dots, x_n) \in \mathbb{F}[x_1, \dots, x_n]$ that can be computed by an arithmetic circuit from \mathcal{A} . We are allowed to ask for the value of the polynomial at points of our choice² and we would like to output a succinct representation for it. Ideally we would like to output an arithmetic circuit from \mathcal{A} that computes the polynomial. This problem is also known as the polynomial interpolation problem.

Unlike the exact learning scenario, there are almost no results that show the impossibility of interpolating arithmetic circuits. To the best of our knowledge, the only hardness result is by [FK06] that proved that polynomial time interpolation of arithmetic circuits implies a lower bound for the class $ZPEXP^{\text{RP}}$. A probable explanation for lack of stronger hardness results is that no reasonable notion of *pseudo-random polynomials* is known in the algebraic domain. However, it is widely believed that analogously to the exact learning case, it is impossible to efficiently interpolate arithmetic circuits of a certain constant depth. The reason for this belief is that depth-3 arithmetic circuits can compute the arithmetic analog of threshold functions (see e.g. [SW01]), and as efficient exact learning of TC_4^0 is believed to be impossible, we also expect efficient interpolation of bounded depth arithmetic circuits to be impossible. It is natural to ask then, what is the maximal depth for which efficient interpolation is possible.

Similarly to the exact learning version, a lot of effort was invested in trying to interpolate restricted classes of arithmetic circuits. In particular, the class of depth-2 arithmetic circuits³ received a great deal of attention and several interpolation algorithms were devised for it [BOT88, GKS94, KS96, Man95, SS96, KS01]. Many works also focused on circuits that can be represented by small *multiplicity automata* [BBV96, BBTV97, BBB⁺00, KS06] and on the class of read-once arithmetic formulae [HH91, BHH95, BC98, BB98, SV08]. One unifying

²In the case of finite fields we may ask for the value over an algebraic extension field of \mathbb{F} .

³Polynomials computed by small depth-2 circuits are also known as sparse polynomials, i.e. polynomials with a small number of monomials.

feature of all these classes is that they all compute polynomials whose partial derivatives span a low dimensional space (see e.g. [KS06] where depth-3 circuits of a very special form are discussed). In contrary, it is easy to give an example of a multilinear $\Sigma\Pi\Sigma(2)$ circuit that computes a polynomial whose partial derivatives span a high dimensional space. Thus, known techniques cannot give efficient algorithms for interpolating polynomials computed by multilinear $\Sigma\Pi\Sigma(2)$ circuits. This highlights the gap in our understanding of depth-2 circuits and depth-3 circuits (even those with only two multiplication gates).

Thus, current techniques are incapable of interpolating depth-3 circuits, even those with two multiplication gates, and it is believed that above some constant depth efficient interpolation is impossible.

In this work we introduce new techniques that enable us to give interpolation algorithms to the class of depth-3 circuits with two multiplication gates. Before presenting our results we need to give several definitions.

1.3 Some definitions and statement of our results

Let f be a polynomial computed by a $\Sigma\Pi\Sigma(2)$ circuit. Then f has the following form:

$$f(\bar{x}) = \prod_{i=1}^{d_1} L_i^{(1)}(\bar{x}) + \prod_{i=1}^{d_2} L_i^{(2)}(\bar{x}), \quad (1)$$

where the $L_i^{(j)}$ -s are linear functions in the variables $\bar{x} = (x_1, \dots, x_n)$, over the field \mathbb{F} :

$$L_i^{(j)}(\bar{x}) = \sum_{k=1}^n \alpha_{i,j,k} x_k + \alpha_{i,j,0}$$

for $\alpha_{i,j,k} \in \mathbb{F}$. Let M_1 and M_2 be the multiplication gates of the circuit. That is,

$$M_1 = \prod_{i=1}^{d_1} L_i^{(1)}(\bar{x}) \quad \text{and} \quad M_2 = \prod_{i=1}^{d_2} L_i^{(2)}(\bar{x}).$$

For a $\Sigma\Pi\Sigma(2)$ circuit C we denote with $\deg(C)$ the maximal degree of its multiplication gates. For example, if C is given by equation (1) then $\deg(C) = \max(d_1, d_2)$. Our first result deals with the case of multilinear $\Sigma\Pi\Sigma(2)$ circuits. A multilinear circuit is a circuit in which every multiplication gate computes a multilinear polynomial. In particular the degree of a multilinear circuit is bounded by n .

Theorem 1. *Let f be a multilinear polynomial in n variables that is computed by a degree d multilinear $\Sigma\Pi\Sigma(2)$ circuit, over a field \mathbb{F} . Then there is a randomized interpolation algorithm that given black box access to f and the parameters d and n runs in $\text{poly}(n, |\mathbb{F}|)$ -time and with probability $1 - o(1)$ outputs all the $\Sigma\Pi\Sigma(2)$ circuits computing f . When $|\mathbb{F}| < n^5$ the algorithm is allowed to make queries to f from a polynomial size algebraic extension field of \mathbb{F} .*

Notice that we output all the multilinear $\Sigma\Pi\Sigma(2)$ circuits computing f . While this was not our purpose when first studying the problem it is a nice consequence of our techniques (and in particular implies that there are only polynomially many such circuits).

In order to state our main theorem we need some more definitions. Let C be as in Equation 1. Define

$$\gcd(C) \triangleq g.c.d.(M_1, M_2)$$

as the greatest common divisor of the multiplication gates. It is clear that we can write $\gcd(C) = \prod_{i=1}^k L_i(\bar{x})$, for some set of linear functions. Following the notations of [DS06] we define the simplification of C , $\text{sim}(C)$, to be the circuit

$$\text{sim}(C) \triangleq C / \gcd(C).$$

From the definition of $\text{sim}(C)$ it is clear that there exists a subset⁴ $I_1 \subseteq [d_1]$ and a subset $I_2 \subseteq [d_2]$, such that $|I_1| = d_1 - k$, $|I_2| = d_2 - k$ and

$$\text{sim}(C) = \prod_{i \in I_1} L_i^{(1)}(\bar{x}) + \prod_{i \in I_2} L_i^{(2)}(\bar{x}). \quad (2)$$

We shall also need the notions of the rank of a $\Sigma\Pi\Sigma(2)$ circuit, which we denote with $\text{rank}(C)$, and of *depth-3 rank* of f , which we denote with $\text{rank}(f)$. Given a $\Sigma\Pi\Sigma(2)$ arithmetic circuit C , let $\text{sim}(C)$, I_1 and I_2 be as in equation (2). We define

$$\text{rank}(C) \triangleq \dim \left(\text{span} \left\{ L_i^{(1)}, L_j^{(2)} : i \in I_1, j \in I_2 \right\} \right).$$

In other words, the rank of C is defined to be the dimension of the space spanned by the linear functions in $\text{sim}(C)$. Let $\text{rank}_d(f)$ be the minimum, over all $\Sigma\Pi\Sigma(2)$ circuits C of degree $\leq d$ that compute f , of $\text{rank}(C)$. The motivation for the definition will become clearer in the analysis of our algorithm. When the degree d is clear from the context then we drop the subscript d and simply write $\text{rank}(f)$ instead of $\text{rank}_d(f)$.

We can now state our second result that deals with general $\Sigma\Pi\Sigma(2)$ circuits.

Theorem 2. *Let f be an n -variate polynomial computed by a $\Sigma\Pi\Sigma(2)$ circuit of degree d , over a field \mathbb{F} . Then there is a randomized interpolation algorithm that given black box access to f and the parameters d and n runs in quasi-polynomial time (in $n, d, |\mathbb{F}|$) and has the following properties:*

- *If $\text{rank}(f) = \Omega(\log^2(d))$, then with probability $1 - o(1)$ the algorithm outputs the (unique) $\Sigma\Pi\Sigma(2)$ circuit for f .*
- *If $\text{rank}(f) = O(\log^2(d))$, then the algorithm outputs, with probability $1 - o(1)$, a polynomial $\text{Lin}(f)$, a polynomial $Q(y_1, \dots, y_k)$ and k linear functions L_1, \dots, L_k , where $k \leq \text{rank}(f)$, such that $\text{Lin}(f)$ is a the product of all the linear factors of f and $\text{Lin}(f) \cdot Q(L_1, \dots, L_k) = f$.*

⁴As usual, $[k]$ stands for the set $\{1, \dots, k\}$.

When $|\mathbb{F}| < \max(d^5, n^5)$ the algorithm is allowed to make queries to f from a polynomial size extension field of \mathbb{F} .

We note that in the case that $\text{rank}(f) = O(\log^2(d))$ the polynomial $Q(L_1, \dots, L_k)$ can be easily represented as a $\Sigma\Pi\Sigma$ circuit with quasi-polynomially many multiplication gates (as Q has at most a quasi-polynomial number of monomials).

1.4 Our techniques

Our algorithms are based on the following scheme. We first restrict the inputs to the unknown polynomial to a low dimensional random subspace of \mathbb{F}^n (although in the multilinear case the subspace is not completely random, the intuition is the same). We then interpolate the polynomial on this subspace. The next step is to lift the representation that we found to the whole space. While this is the general scheme it has different realizations in the multilinear case and the general case, and even within each case we have to deal differently with the case of high rank and the case of low rank. However, similar problems lie in the core of the different cases. The following questions give a good intuition to the difficulties that the algorithm has to overcome.

1. Let V_1, \dots, V_k be subspaces of co-dimension 1 inside a linear space V . Given circuits C_1, \dots, C_k such that C_i computes $f|_{V_i}$ (the restriction of f to V_i) how can we construct from them a single circuit C for $f|_V$?
2. Given linear spaces $V \subseteq U$ such that V is of co-dimension 1 in U , and a circuit C computing $f|_V$, how many circuits C' are there that compute $f|_U$ and whose restriction $C'|_V$ is equal to C ?

The first question arises in the case of general $\Sigma\Pi\Sigma(2)$ circuits of high rank when we interpolate the restriction of f to a random subspace V . Our algorithm first interpolates the restrictions of one of the multiplication gates to co-dimensional 1 subspaces of V and then combines the different results to get a representation of that gate over V (then using the factoring algorithms of [Kal85, KT90, Kal95] we are able to interpolate $f|_V$).

To deal with the problem we consider linear functions in the different C_i -s that look similar to each other and try to glue them together to get a new function. This process may fail if for any linear function in, say, C_1 there are many other linear functions in C_1 that are at Hamming distance 1 from it. In such a situation it is hard for us to tell what is the “true” image of that linear function in the other C_i -s. On the other hand, if this is indeed the case then the linear functions in C_1 generate a *locally decodable code* and using the results of [GKST06, DS06] on the length of such codes, we can prove that such anomaly cannot happen. Therefore we can always find a linear function to learn, until eventually we find the whole multiplication gate.

The motivation for the second question is that when we lift the representation that we found over V to U there may be many different circuits that are possible lifts, and we

somehow have to pick the right one to continue the lifting process with. To deal with this problem we note that if a polynomial has two different lifts then the difference of the lifts is the zero polynomial. By a result of [DS06] regarding the structure of identically zero depth-3 circuits, we get that the different lifts must be of small rank. This enables us to solve the problem for the high rank case (as in this case the lift is unique). The low rank case is indeed more problematic and this is why we need to output a circuit with quasi-polynomially many multiplication gates when f has low rank (although in the multilinear case we manage to overcome this difficulty by proving that the total number of possible lifts is polynomial).

1.5 Recent progress

In a recent joint work with Z. Karnin [KS08] we managed to extend this result to the case of $\Sigma\Pi\Sigma(k)$ circuits. Moreover, we managed to get a deterministic reconstruction algorithm (the algorithms in this paper are all randomized). The ideas in the current paper serve as a basic building block to the ideas of [KS08], however some new ideas were required for the more general case.

1.6 Organization

The paper is organized as follows. In Section 2 we give some definitions and describe the results of [DS06] regarding identically zero depth-3 circuits. The algorithm and proof for the multilinear case is given in Section 3. In Section 4 we give the algorithm (and proof) for the general case.

2 Preliminaries

For a natural number n we denote $[n] = \{1, \dots, n\}$. For a set $S \subset [n]$ we denote with \bar{S} the complement of S .

Let \mathbb{F} be a field. We denote with \mathbb{F}^n the n -dimensional vector space over \mathbb{F} . We shall use the notation $\bar{x} = (x_1, \dots, x_n)$ to denote the vector of n indeterminates. For $v \in \mathbb{F}^n$ we denote with $\text{wt}(v)$ the weight of v , i.e. the number of non zero coordinates in v . For two non-zero linear functions L_1, L_2 we will write $L_1 \sim L_2$ whenever L_1 and L_2 are linearly dependent. Let $V = V_0 + v_0 \subseteq \mathbb{F}^n$ be an affine subspace, where $v_0 \in \mathbb{F}^n$ and $V_0 \subseteq \mathbb{F}^n$ is a linear subspace. Let $L(\bar{x})$ be a linear function. We denote with $L|_V$ the restriction of L to V . We say that a set of linear functions $\{L_1, \dots, L_k\}$ is linearly independent over V if the only linear combination of the L_i -s whose restriction to V is identically zero is the all zero combination. We now introduce coordinates to the space V . Let v_1, \dots, v_s be a basis of V_0 . Let $L(\bar{x})$ be a linear function. We consider the restriction of L to V with respect to the basis $\{v_i\}$. Then $L|_V$ can be written as a function in s variables. In particular if $v = \alpha_1 v_1 + \dots + \alpha_s v_s + v_0$ then we define $L|_V(\alpha_1, \dots, \alpha_s) = L(v) = L(\alpha_1 v_1 + \dots + \alpha_s v_s) + L(v_0)$.

For a polynomial f we denote with $\text{Lin}(f)$ the product of all the linear factors of f (with the appropriate multiplicities). We also define $\text{sim}(f) = f/\text{Lin}(f)$ to be the simplification of f . Clearly $\text{sim}(f)$ does not have any linear factors.

2.1 Identically zero depth-3 circuits

In this section we state some results of [DS06] regarding identically zero depth-3 circuits. We start by giving some necessary definitions. A $\Sigma\Pi\Sigma(k)$ circuit (that is, a depth-3 circuit with k multiplication gates) is identically zero if it computes the zero polynomial. Notice that this is a syntactic definition, we are thinking of the circuit as computing a polynomial and not a function over the field.⁵ Let $C = M_1 + \dots + M_k$ be an identically zero $\Sigma\Pi\Sigma(k)$ circuit, where the M_i -s are the multiplication gates. We say that C is minimal if there is no $\emptyset \neq I \subsetneq [k]$ such that $\sum_{i \in I} M_i \equiv 0$. C is simple if the g.c.d. of its multiplication gates is 1. The following theorem gives a bound on the degree of multilinear $\Sigma\Pi\Sigma(k)$ circuits that are identically zero.

Theorem 3 (Corollary 6.9 of [DS06]). *There exists an integer function $D(k) = 2^{O(k^2)}$ such that every simple, minimal, identically zero multilinear $\Sigma\Pi\Sigma(k)$ circuit is of degree $d \leq D(k)$.*

In other words, if C is an identically zero $\Sigma\Pi\Sigma(k)$ multilinear circuit that is simple and minimal, then the degree of C is bounded by a constant depending on k . We will need to use the result only for $\Sigma\Pi\Sigma(4)$ circuits. We state this as a corollary, and introduce the constant D_4 that will play a part in our interpolation algorithms.

Corollary 4. *There exists a constant D_4 , such that every identically zero multilinear $\Sigma\Pi\Sigma(4)$ circuit that is simple and minimal is of degree $\leq D_4$.*

The next theorem deals with general $\Sigma\Pi\Sigma(k)$ circuit.

Theorem 5 (Lemma 5.2 of [DS06]). *Let $k \geq 3$, $d \geq 2$ be integers and $C \equiv 0$ be a simple and minimal $\Sigma\Pi\Sigma(k)$ circuit. Then $\text{rank}(C) \leq 2^{O(k^2)} \log^{k-2}(d)$.*

As before we shall need the following corollary and the constant R_4 .

Corollary 6. *There exists a constant R_4 , such that every identically zero $\Sigma\Pi\Sigma(4)$ circuit, of degree d , that is simple and minimal is of rank at most $R_4 \cdot \log^2(d)$.*

The following corollary shows how to use Corollaries 4 and 6 to guarantee uniqueness of a $\Sigma\Pi\Sigma(2)$ circuit.

Corollary 7. *Let d be some integer and D_4, R_4 as in Corollaries 4, 6, respectively. If a polynomial f has a degree d (multilinear) $\Sigma\Pi\Sigma(2)$ circuit and $\text{rank}(f) > R_4 \cdot \log^2(d)$ ($\text{deg}(f) > D_4$) then f has a unique (multilinear) $\Sigma\Pi\Sigma(2)$ circuit of degree d .*

Proof. We prove the claim only for general $\Sigma\Pi\Sigma(2)$ circuits. The proof for the multilinear case is identical. Let $C_1 = A_1 + A_2$ be a $\Sigma\Pi\Sigma(2)$ circuit for f , where A_1, A_2 are its

⁵In our case the field size is much larger than the degree of the polynomial and so the syntactic definition and the semantic one are the same. However, we mention this distinction as it is sometime a cause of confusion.

multiplication gates. By our assumption on $\text{rank}(f)$ we know that

$$\text{rank}(C_1/\text{g.c.d.}(A_1, A_2)) > R_4 \cdot \log^2(d).$$

Assume that $C_2 = B_1 + B_2$ is another $\Sigma\Pi\Sigma(2)$ circuit for f , where B_1, B_2 are its multiplication gates. Consider the circuit $C = A_1 + A_2 - B_1 - B_2$, then C is a $\Sigma\Pi\Sigma(4)$ circuit, and $C \equiv 0$. By the assumption on $\text{rank}(C_1)$ we get that $\text{rank}(C) > R_4 \cdot \log^2(d)$. By Corollary 6 (for the case of multilinear circuits we use Corollary 4) this implies that C is either not simple or not minimal. Note, that by the assumption on $\text{rank}(C_1/\text{g.c.d.}(A_1, A_2))$ we get that even if we remove the g.c.d. of C the remaining circuit still has $\text{rank} > R_4 \cdot \log^2(d)$. From this we conclude that C is not minimal. As $C_1 \not\equiv 0$ we get that $A_1 - B_1 \equiv 0$ or $A_1 - B_2 \equiv 0$. \square

3 Multilinear circuits

In this section we prove Theorem 1. To ease the reading we repeat it here.

Theorem 1 Let f be a multilinear polynomial in n variables that is computed by a degree d multilinear $\Sigma\Pi\Sigma(2)$ circuit, over a field \mathbb{F} . Then there is a randomized interpolation algorithm that given black box access to f and the parameters d and n runs in $\text{poly}(n, |\mathbb{F}|)$ -time and with probability $1 - o(1)$ outputs *all* the $\Sigma\Pi\Sigma(2)$ circuits computing f . When $|\mathbb{F}| < n^5$ the algorithm is allowed to make queries to f from a polynomial size algebraic extension field of \mathbb{F} .

Before sketching the proof we repeat some of the notations that we will use. We shall have the following representation of f in mind: $f = M_1 + M_2$ where M_1 and M_2 are the multiplication gates of a multilinear $\Sigma\Pi\Sigma(2)$ circuit for f that are given by the equations

$$M_1 = \prod_{i=1}^d L_i(S_i) \quad \text{and} \quad M_2 = \prod_{j=1}^{d'} L'_j(S'_j), \quad (3)$$

where the L_i -s (L'_j -s) are linear functions and the sets $\{S_i\}_{i \in [d]}$ ($\{S'_j\}_{j \in [d']}$) form a partition of the set of variables (recall that M_1 and M_2 compute multilinear polynomials). In particular

$$f(\bar{x}) = \prod_{i=1}^d L_i(S_i) + \prod_{j=1}^{d'} L'_j(S'_j). \quad (4)$$

To prove the theorem we will give an algorithm for reconstructing all the $\Sigma\Pi\Sigma(2)$ circuits for f and prove its correctness. The algorithm basically follows the scheme that was described in Section 1.4. We have two cases, the low rank case and the high rank case (in fact we look at low degree and high degree, but for multilinear circuits this corresponds to low rank and high rank). More precisely, in the multilinear setting low rank means constant rank (where

the constant depends on D_4 as defined in Corollary 4). This makes life easier compared to the general case that we will study in Section 4 in which low rank can be as large as $O(\log^2(d))$. We now give a slightly more detailed sketch of the proof.

- **Preprocessing:** The first step in the proof is a preprocessing step in which we find all the linear factors of f . We then show that in the multilinear case each linear factor of f is also a linear factor of both multiplication gates in every $\Sigma\Pi\Sigma(2)$ circuit for f . This enables us to reduce the problem of finding all the circuits for f to finding the circuits for $\text{sim}(f)$. After this step the algorithm treats separately low degree circuits and high degree circuits.
- **Low degree circuits:** The idea is to consider all the restrictions of $\text{sim}(f)$ to small subsets of the variables. Namely, for a set S the restriction of $\text{sim}(f)$ to S is done by setting the variables not in S to zero. We denote the restriction of $\text{sim}(f)$ to S with $\text{sim}(f)(S)$. We will only consider sets S of small size and so we can find (in a brute force manner) all the simple multilinear $\Sigma\Pi\Sigma(2)$ circuits for $\text{sim}(f)(S)$. Then we will prove that for every multilinear $\Sigma\Pi\Sigma(2)$ circuit C for $\text{sim}(f)$ there exists a set S and a simple multilinear circuit A on the variables in S such that A is the restriction of C to S (which is defined in a similar manner to $\text{sim}(f)(S)$) and that there is a relatively easy way of reconstructing C from A by “revealing” each variable separately.
- **High degree circuits:** The algorithm for the high degree case is similar in spirit to the low degree case with a few exceptions. The first difference is that we will not restrict the variables not in S to zero but rather substitute random values from \mathbb{F} to each of them. The second difference is that in the low degree case the degree of the circuit A (that was described in the previous item) is the same as the degree of C , whereas in the high degree case after restricting the circuit to a small set the degree must drop (it can be at most $|S|$ as the circuit is multilinear).

3.1 Preprocessing step

Before describing the algorithms for the low rank and high rank cases we have a preprocessing step in which we find all the linear factors of f . We will show that in the case of multilinear $\Sigma\Pi\Sigma(2)$ circuits we actually get that the linear factors of f are also linear factors of each multiplication gate separately. This allows us to reduce the case of reconstructing multilinear $\Sigma\Pi\Sigma(2)$ circuits to the problem on interpolating multilinear $\Sigma\Pi\Sigma(2)$ circuits that do not have linear factors. We start by showing that we can find the linear factors efficiently. The following theorem is an immediate corollary of the results of [Kal85, KT90, Kal95]. The theorem requires that the field that we are working with is not too small so from now on we shall assume that $|\mathbb{F}| \geq n^5$ (we can make this assumption as we are allowed to query f on inputs from an extension field).

Theorem 8. *Let d, n be integers. Let \mathbb{F} be a finite field. Then there is a randomized algorithm A that gets as input a black box access to f and the parameters n and d , and*

outputs, in $\text{poly}(n, d, \log |\mathbb{F}|)$ time, with probability $1 - \exp(-n)$, all the linear factors, with their multiplicities, of f .

Let $\text{Lin}(f)$ be the product of all the linear factors of f . The following lemma shows that if f is not a product of linear functions then every linear function in its factorization also divides the multiplication gates M_1 and M_2 (note that this is not the case for non multilinear $\Sigma\Pi\Sigma(2)$ circuits⁶).

Lemma 9. *Let f be a polynomial that is computable by a multilinear $\Sigma\Pi\Sigma(2)$ circuit (as in Equation (4)). Assume that f cannot be represented as a product of linear functions. Then if a linear function L divides f then L must also divide both multiplication gates, in any multilinear $\Sigma\Pi\Sigma(2)$ circuit for f .*

Proof. Consider a multilinear $\Sigma\Pi\Sigma(2)$ circuit for f . Assume w.l.o.g. that it is given by Equation (4). As we assume that f cannot be represented as a product of linear functions we can also assume w.l.o.g. that $\text{g.c.d.}(M_1, M_2) = 1$. Assume for a contradiction that L does not divide M_1 (and therefore does not divide M_2). Consider the (affine) subspace on which $L = 0$. We must have that $f|_{L=0} = 0$. This implies that $M_1|_{L=0} = -M_2|_{L=0}$. As M_1 and M_2 are both products of linear functions we get that they share the same linear factors modulo L . In particular we can assume w.l.o.g. that⁷ $d' = d$ and $L_i|_{L=0} \sim L'_i|_{L=0}$. By examination we get that the support of L (that is the set of non zero coordinates of L) is contained in the union of the supports of L_i and L'_i , for every $1 \leq i \leq d$ (recall that we assume that $\text{g.c.d.}(M_1, M_2) = 1$). If $d > 2$ then this is not possible as the circuit for f is multilinear and the S_i -s are disjoint (and so are the S'_j -s). Thus we get that $d = 2$ (recall that we assumed that we removed the g.c.d.). However, if $d = 2$ and L divides f then there is another linear function L' such that $f = L \cdot L'$ in contradiction to the assumption that f is not a product of linear functions. \square

Thus, in contrary to the general case (i.e. non-multilinear circuits), every linear factor of f is also a linear factor of both multiplication gates in every multilinear $\Sigma\Pi\Sigma(2)$ circuit for f (unless f itself is a product of linear functions). Recall that $\text{sim}(f) = f/\text{Lin}(f)$. Thus, if we have $\text{Lin}(f)$ at hand then it is easy to simulate oracle access to $\text{sim}(f)$ by making queries to f .⁸ Moreover, as we assume that we are given the degree d of a multilinear $\Sigma\Pi\Sigma(2)$ circuit for f then it is easy to compute $D \triangleq d - \deg(\text{Lin}(f))$. Hence, in the rest of the section we will reconstruct all the simple multilinear $\Sigma\Pi\Sigma(2)$ circuits for $\text{sim}(f)$ of degree D , and from them we will immediately get all the multilinear $\Sigma\Pi\Sigma(2)$ circuits for f .

⁶E.g. consider the circuit $(x+w)(x+y)(x+z) - wyz$. It is not hard to see that it is divisible by x , but cannot be represented as a product of linear functions.

⁷We can also have $d = d' \pm 1$ but the analysis remains the same.

⁸Given $\text{Lin}(f)$ and oracle access to f and a point $\alpha \in \mathbb{F}^n$ we would like to output $\text{sim}(f) = f(\alpha)/\text{Lin}(f)(\alpha)$. There may be a problem when $\text{Lin}(f)(\alpha) = 0$ however this can be easily taken care of by passing a line through α that contains enough (e.g. more than d) points on which $\text{Lin}(f)$ does not vanish. This is a routine procedure and so we omit its details here.

As described in the sketch we have two cases. The case that⁹ $D \leq D_4$ (low degree case) and the case that $D > D_4$ (high degree case). We begin with the low degree case.

3.2 Multilinear circuits: Low degree case

In this section we give an algorithm that finds all the multilinear $\Sigma\Pi\Sigma(2)$ circuits of degree $D \leq D_4$ that compute $\text{sim}(f)$. We assume w.l.o.g. that the number of variables in $\text{sim}(f)$ is more than $10D_4$ as otherwise we can find all the circuits for $\text{sim}(f)$ by a brute force search (this assumption is used in Lemma 11).

We start with a sketch of the algorithm. We first compute the sum-product representation of f . Note that as the degree of $\text{sim}(f)$ is at most D_4 then we can interpolate $\text{sim}(f)$ in polynomial time. This will be helpful in future steps where we have to verify that a given circuit computes a certain restriction of $\text{sim}(f)$. In the next step, for each subset S of the variables of size $|S| = 10D_4$, we consider the restriction of $\text{sim}(f)$ in which every variable not in S is set to zero. We denote this polynomial with $\text{sim}(f)(S)$. We then find, in a brute force manner, all the degree $\leq D$ multilinear $\Sigma\Pi\Sigma(2)$ circuits that compute $\text{sim}(f)(S)$. We now wish to construct all the circuits computing $\text{sim}(f)$ from the circuits for the different $\text{sim}(f)(S)$ -s. To do so, we consider for every such set S all the sets of the form $S \cup \{i\}$ for $i \notin S$. For each such new set we again compute all the circuits for $\text{sim}(f)(S \cup \{i\})$. Now we try to combine circuits for $\text{sim}(f)(S \cup \{1\}), \dots, \text{sim}(f)(S \cup \{n\})$ to a single circuit for $\text{sim}(f)$. At first sight it may not be clear how to do this combination, however we prove that for every circuit for $\text{sim}(f)$ there is a set S for which such a combination will be easy to find. We now give a more formal description of the algorithm (Algorithm 1).

Algorithm 1 shows how to find all the multilinear $\Sigma\Pi\Sigma(2)$ circuits that compute $\text{sim}(f)$, when its degree D is at most D_4 . As we are looking for multilinear circuits and we have $\text{Lin}(f)$ we shall only consider variables that do not belong to $\text{Lin}(f)$. In order not to add further notations we shall assume that the variables appearing in $\text{sim}(f)$ are $\{x_1, \dots, x_n\}$.

Before giving the analysis of the algorithm we explain in what way does it follow the scheme described in Section 1.4. Recall that according to the sketch in 1.4 we first have to restrict $\text{sim}(f)$ to a random subspace of the inputs. As restriction to a subspace does not preserve multilinearity (imagine the polynomial xy restricted to the subspace $x = y$) we only consider subspaces in which some of the variables are restricted to zero. Clearly a restriction to such a subspace preserves multilinearity. Indeed, in Step 2 we basically interpolate all the restrictions of $\text{sim}(f)$ to subspaces of this form that have a low dimension. The lifting is preformed in Step 4.

The following theorem summarizes the required properties of Algorithm 1.

Theorem 10. *Let f be a multilinear polynomial that can be computed by a multilinear $\Sigma\Pi\Sigma(2)$ circuit. Assume that $\deg(\text{sim}(f)) > 3$. Then algorithm 1, when given oracle access to $\text{sim}(f)$, outputs all the multilinear $\Sigma\Pi\Sigma(2)$ circuits of degree D computing $\text{sim}(f)$. The*

⁹Recall that D_4 is defined in Corollary 4.

Algorithm 1 Multilinear circuits of low degree

1. Interpolate the polynomial $\text{sim}(f)$ to get an explicit representation of it as a sum of monomials.
 2. $\forall S \subseteq [n]$ of size $|S| = 10D_4$ find all the degree D *simple* multilinear circuits in the variables of S that compute $\text{sim}(f)(S)$ (recall that $\text{sim}(f)(S)$ is the polynomial obtained after setting the variables not in S to zero).
 3. For each such set S and circuit A computing $\text{sim}(f)(S)$ do the following: for $i \notin S$ set $S_i = S \cup \{i\}$. Repeat the previous steps and find all the multilinear $\Sigma\Pi\Sigma(2)$ circuits A_i that compute $\text{sim}(f)(S_i)$ and such that $A_i|_{x_i=0} \equiv A$ (i.e. after substituting $x_i = 0$ the circuits are identical). If for some i there is no such circuit then move to the next circuit A for $\text{sim}(f)(S)$.
 4. For each S and A for which we found $\{A_i\}_{i \in \bar{S}}$ combine, if possible, the different circuits into one multilinear $\Sigma\Pi\Sigma(2)$ circuit: for every linear function $L \in A$ let $L_i \in A_i$ be such that $L_i|_{x_i=0} = L$. Denote $L_i = L + \alpha_i \cdot x_i$. Replace L with $\tilde{L} = L + \sum_{i \notin S} \alpha_i \cdot x_i$.
 5. For each circuit found in the previous step verify that it computes $\text{sim}(f)$.
-

running time of the algorithm is polynomial in n and $|\mathbb{F}|$.

Note that the theorem only discusses polynomials f such that $\text{sim}(f)$ has degree larger than 3. This is because when the degree of $\text{sim}(f)$ is too small then there may be more possible circuits computing it. We deal with this case in Section 3.2.1.

Proof. Before proving the correctness of the algorithm we first analyze its running time. The first step in the algorithm is a simple interpolation of a multilinear polynomial of degree D in (at most) n variables. This step runs in time polynomial in n^D (we simply query the polynomial on all inputs in $\{0, 1\}^n$ of weight at most D , and solve a system of linear equations to find the coefficients, remembering to look only at coefficients of monomials of degree at most D). Hence we can assume that we have an explicit representation of $\text{sim}(f)$ as sum of monomials. It is clear that in the second step we get a polynomial number of circuits. Indeed there are at most n^{10D_4} many sets S , and for each set S there are at most $|\mathbb{F}|^{2D \cdot (10D_4+1)}$ multilinear $\Sigma\Pi\Sigma(2)$ circuits in the variables of S (we just count the number of sets of at most $2D$ linear functions, in $10D_4$ variables). In the same way we see that computing the different A_i -s also requires polynomial time. As we have a description of $\text{sim}(f)$ at our disposal it is easy to verify whether a given circuit computes $\text{sim}(f)(S)$. Thus, Steps 1-3 can be computed in polynomial time (in $|\mathbb{F}|$ and n). It is also clear that Step 4 requires a polynomial time, as the number of circuits that we computed in the previous steps is also polynomial. The last step (verification step) also requires a polynomial running time hence the total running time is polynomial in n and $|\mathbb{F}|$.

The correctness of the algorithm follows from the next two lemmas. The first lemma (Lemma 11) shows that there exists a set S such that $\text{sim}(f)(S)$ does not have linear factors, and hence Step 2 can be implemented for $\text{sim}(f)$. The second lemma (Lemma 12) shows that if S satisfies a certain property and A computes $\text{sim}(f)(S)$ then for every $i \notin S$ there is a unique circuit A_i that computes $\text{sim}(f)(S \cup \{i\})$ and that equals A after setting $x_i = 0$.

Lemma 11. *Let $g(x_1, \dots, x_n)$ be a polynomial on $n > 2d+1$ variables (recall our assumption at the beginning of Section 3.2), that has no linear factors and that can be computed by a $\Sigma\Pi\Sigma(2)$ circuit of degree d . Then there exists a variable x_k such that $g|_{x_k=0}$ has no linear factors.*

Proof. Let $C = \prod_{i=1}^{d_1} L_i + \prod_{i=1}^{d_2} L'_i$ be some $\Sigma\Pi\Sigma(2)$ circuit computing g , for some $d_1, d_2 \leq d$. It is clear that $\dim(\text{span}(\{1, L_1, \dots, L_{d_1}, L'_1, \dots, L'_{d_2}\})) \leq 2d + 1$. In particular there is $1 \leq i \leq n$ such that x_i is not spanned by the linear functions in the circuit (and the constant function). Assume w.l.o.g that $i = n$. It is now clear that if we set x_n to zero then the resulting circuit C' will be “isomorphic” to C in the following sense: there exists an invertible linear transformation $\phi : \mathbb{F}^n \rightarrow \mathbb{F}^n$ such that for every $(a_1, \dots, a_n) \in \mathbb{F}^n$ we have that $C(a_1, \dots, a_n) = C'(\phi(a_1, \dots, a_n))$ (we think of C' as a circuit in n variables although x_n does not appear in it). In particular if C' has a linear factor then so does C (as C is equal to the composition of C' with a linear transformation). As C has no linear factors (by our assumption on g) we get that $C' = C|_{x_n=0}$ has no linear factors as well. \square

Note that the lemma works for any $\Sigma\Pi\Sigma(2)$ circuit and not just multilinear circuits. We also note that it is not difficult to change the proof of the lemma to hold also for the case that $n = 2d + 1$ (we just need to consider the homogeneous part of each L_i and L'_j and not consider the constant function 1). In addition, it is also easy to see that we cannot have $n = 2d$ because of the following example $f(x_1, \dots, x_{2d}) = \prod_{i=1}^d x_i + \prod_{i=d+1}^{2d} x_i$. The next lemma shows that if $D > 3$ then for every S_i and circuit A that was computed in Step 2 there is at most one multilinear $\Sigma\Pi\Sigma(2)$ circuit A_i with $A_i = \text{sim}(f)(S_i)$ and $A_i|_{x_i=0} \equiv A$. The proof is by a simple manipulation of polynomials. Note that this implies that if there exists a circuit C for f such that A is its restriction to the variables S (that is, A is the resulting circuit after setting all the variables not in S to zero), then Step 4 will return the circuit C (because of the uniqueness we are assured that we combine the linear functions in the different A_i -s in the only possible way).

Lemma 12. *Let $g(x_1, \dots, x_t)$ be a polynomial of degree $D > 3$ that does not have any linear factors. Assume that we have a multilinear $\Sigma\Pi\Sigma(2)$ circuit computing $g|_{x_t=0}$. In other words we have*

$$g(x_1, \dots, x_t)|_{x_t=0} = \prod_{i=1}^D L_i + \prod_{j=1}^{D'} L'_j.$$

Then there is at most one multilinear circuit of the form

$$\prod_{i=1}^D (L_i + \alpha_i x_t) + \prod_{j=1}^{D'} (L'_j + \alpha'_j x_t)$$

that computes g (note that such a circuit may not exist).

Proof. Assume for a contradiction that there are two different multilinear circuits of that form that compute g . Assume w.l.o.g. that the first circuit is

$$g = C_1 = (L_1 + \alpha x_t) \cdot \prod_{i=2}^D L_i + (L'_1 + \alpha' x_t) \cdot \prod_{j=2}^{D'} L'_j.$$

There are three canonical options for the second circuit

$$C_2 = L_1 \cdot (L_2 + \beta x_t) \cdot \prod_{i=3}^D L_i + L'_1 \cdot (L'_2 + \beta' x_t) \cdot \prod_{j=3}^{D'} L'_j. \quad (5)$$

$$C_2 = (L_1 + \beta x_t) \cdot L_2 \cdot \prod_{i=3}^D L_i + L'_1 \cdot (L'_2 + \beta' x_t) \cdot \prod_{j=3}^{D'} L'_j. \quad (6)$$

$$C_2 = (L_1 + \beta x_t) \cdot L_2 \cdot \prod_{i=3}^D L_i + (L'_1 + \beta' x_t) \cdot L'_2 \cdot \prod_{j=3}^{D'} L'_j. \quad (7)$$

We shall only analyze the first case (given in Equation (5)) as it is the more interesting one. The analysis of the other cases is similar (and somewhat simpler).

So let us assume that C_2 is given by Equation (5). As $C_1 = C_2$ we get, by exchanging sides, that

$$x_t \cdot (\alpha L_2 - \beta L_1) \cdot \prod_{i=3}^D L_i = x_t \cdot (\beta' L'_1 - \alpha' L'_2) \cdot \prod_{j=3}^{D'} L'_j.$$

Since $D > 3$ there is $3 \leq i \leq D$ such that $L_i \nmid (\beta' L'_1 - \alpha' L'_2)$ and so there must be $3 \leq j \leq D$ such that $L_i \sim L'_j$. This implies that L_i divides g . However we assumed that g does not have linear factors, so we have a contradiction. \square

The proof of Theorem 10 now follows easily. Indeed, by Lemma 11 we know that for every circuit C for $\text{sim}(f)$ there exists a set S of size $|S| = 10D_4$ and a simple multilinear circuit A that compute $\text{sim}(f)(S)$. Lemma 12 and the discussion proceeding it show that C will be one of the circuit computed in Step 4. It is clear that the last step will only keep the correct circuits. \square

We now handle the cases of $D = 3$ and $D = 2$ (note that Lemma 12 says nothing for such D 's and indeed the claim is not true in this case).

3.2.1 Case $D \leq 3$

Algorithm 1 in its current form cannot compute all the circuits of degree ≤ 3 for $\text{sim}(f)$. However, the required change is minor and so we describe it here and do not repeat the algorithm itself. We also give a sketch of a proof as the proof itself contains easy manipulations similar in spirit to those that were already performed for the case $D > 3$. The basic difference is in Steps 3 and 4. Instead of computing a circuit A_i for $\text{sim}(f)(S \cup \{i\})$ we need to compute a circuit $A_{i,j}$ for $\text{sim}(f)(S \cup \{i, j\})$. Another change is that it may be the case that for some i the circuit A_i is not the unique lift (in contrast to what Lemma 12 guarantees for higher degrees). However, in such a case we prove that $A_{i,j}$ is unique for every $j \notin S \cup \{i\}$. The rest of the proof should now be clear (given that we prove the above claim regarding $A_{i,j}$). In fact, this discussion is only true for $D = 3$ and for $D = 2$ yet another slight modification is required but it is similar in spirit and so we omit it.

We first analyze the case $D = 3$. The only possible difference from the case $D > 3$ is when there is some index i such that there are two (or more) different ways of extending the circuit A to a circuit A_i . From the proof of Lemma 12 we get that w.l.o.g. A has the following form

$$A = L_1 \cdot L_2 \cdot L_3 + L'_1 \cdot L'_2 \cdot L'_3,$$

and the two different lifts are¹⁰

$$A_i = (L_1 + \alpha_2 x_i) \cdot L_2 \cdot L_3 + (L'_1 - \beta_2 x_i) \cdot L'_2 \cdot L'_3$$

and

$$A'_i = L_1 \cdot (L_2 - \alpha_1 x_i) \cdot L_3 + L'_1 \cdot (L'_2 + \beta_1 x_i) \cdot L'_3,$$

for constant $\alpha_1, \alpha_2, \beta_1$ and β_2 . In particular we must have that $L_3 = c \cdot (\beta_1 L'_1 + c_2 \cdot \beta_2 L'_2)$ and $L'_3 = c \cdot (\alpha_1 L_1 + \alpha_2 L_2)$, for some $c \neq 0$. Using the same reasoning as in the proof of Lemma 12 it is easy to show that for any variable $j \notin S \cup \{i\}$ there is at most one circuit $A_{i,j}(S \cup \{i, j\})$ satisfying

$$A_{i,j} = g(S \cup \{i, j\})$$

and

$$A_{i,j}|_{x_j=0} \equiv A_i.$$

In other words, for the case $D = 3$ we have to perform Steps 3 and 4 of Algorithm 1 for the circuit $A_i(S \cup \{i\})$ and its possible lifts $A_{i,j}(S \cup \{i, j\})$.

To conclude, if we have an index i with two different lifts A_i and A'_i (as described above) then, as in the case of $D > 3$, we get that there are at most two circuits \hat{A} and \hat{A}' that for every $j \notin S \cup \{i\}$ satisfy

$$\hat{A}(S \cup \{i, j\}) \equiv A_{i,j}$$

¹⁰There are two more cases to consider (corresponding to Equations (6) and (7)) but the analysis follows from similar arguments.

and

$$\hat{A}(S \cup \{i, j\}) \equiv A'_{i,j}.$$

Therefore we get, again, that at most polynomially many circuits are found by the algorithm, that those polynomials are computed in polynomial time, and that they contain all the possible representations for $\text{sim}(f)$ (we can find those circuit by repeating Step 4 of Algorithm 1 for each of the A_i -s and the corresponding set $\{A_{i,j}\}$). Hence we can find in polynomial time all the different multilinear $\Sigma\Pi\Sigma(2)$ circuits for f .

The case $D = 2$ follows by a similar case analysis. We omit the proof as this case (as well as the case $D = 3$) is not very interesting and the proofs are simple and resemble the proofs seen so far.

3.3 Multilinear circuits: High degree case

We now turn to the high degree case. As described above, the algorithm for this case is similar to Algorithm 1 with the exception that we do not substitute zeroes to the variables outside S and that “gluing” the different circuits for $\text{sim}(f)(S \cup \{i\})$ is slightly more complicated. Algorithm 2 shows how to interpolate the circuit in case that $\deg(\text{sim}(f)) > D_4$.

Algorithm 2 Multilinear circuits of high degree

$\forall S \subseteq [n]$ such that $|S| = 2D_4$ do the following.

1. $\forall i \notin S$ pick a random assignment to x_i from \mathbb{F} . Denote the final assignment with ρ . Let $\text{sim}(f)|_\rho$ be the polynomial $\text{sim}(f)$ after substituting the partial assignment ρ .
2. For every simple multilinear $\Sigma\Pi\Sigma(2)$ circuit in the variables $\{x_i\}_{i \in S}$, over \mathbb{F} , of *degree greater than D_4* (and at most $2D_4$ of course), check whether the circuit computes $\text{sim}(f)|_\rho$. If no such circuit computing $\text{sim}(f)|_\rho$ is found then move to the next S .
3. For every $i \neq j$, such that $i, j \notin S$ set $S_{i,j} = S \cup \{i, j\}$. Find a simple multilinear $\Sigma\Pi\Sigma(2)$ circuit $A_{i,j}$ that computes $\text{sim}(f)|_{\rho_{i,j}}$, where $\rho_{i,j}$ is the assignment that equals ρ on all the coordinates outside $S_{i,j}$ (namely, we “forget” the assignments to x_i and x_j).
4. Glue the different circuits for the $S_{i,j}$ -s to a single multilinear $\Sigma\Pi\Sigma(2)$ circuit. That is, find the unique circuit C whose restriction to $\rho_{i,j}$ equals $A_{i,j}$ (the exact way of gluing will be explained in the analysis of the algorithm).

If no circuit is found then output “fail”.

The basic intuition behind the algorithm is the following. As we will see in Lemma 15 there is a unique circuit for $\text{sim}(f)$ of degree greater than D_4 (given that one exists). Moreover, we will see that w.h.p. even when we substitute random field elements to most of the variables, there is still a unique circuit for the restricted polynomial. Thus, in order

to find the circuit for $\text{sim}(f)$ we first find the circuit for the restricted polynomial $\text{sim}(f)|_\rho$ (Steps 1 and 2) and then find the unique way of recovering the other variables. The main difference from the low degree case is that by exposing the other variables new linear functions may appear (as the degree of the restricted polynomial is at most $|S|$) and in order to find the partition of the other variables to the new linear functions we have to consider pairs of variables (i.e. find the circuit for $\text{sim}(f)(S \cup \{i, j\})$ instead of just finding a circuit for $\text{sim}(f)(S \cup \{i\})$). This is done in Steps 3 and 4. The following theorem shows that Algorithm 2 is indeed correct and analyze its running time .

Theorem 13. *Let f be a multilinear polynomial such that $\text{sim}(f)$ can be computed by a multilinear $\Sigma\Pi\Sigma(2)$ circuit of degree $D > D_4$. Then algorithm 2, when given oracle access to $\text{sim}(f)$, outputs with probability $\geq 1 - (n + 1)^2/|\mathbb{F}|$, the unique multilinear $\Sigma\Pi\Sigma(2)$ circuits of degree D computing $\text{sim}(f)$. The running time of the algorithm is polynomial in n and $|\mathbb{F}|$.*

Note that in particular the theorem implies that there is a unique degree D $\Sigma\Pi\Sigma(2)$ circuit for $\text{sim}(f)$. This is however a simple fact given Corollary 7 (and in Lemma 15 we prove a slightly more general version of this fact).

Proof. The idea of the proof is the following. We first show (Lemma 14) that for every set S , w.h.p. over the choice of ρ , there is no linear factor dividing $\text{sim}(f)|_\rho$ (i.e. no new linear factor was introduced). We then show (Lemma 15) that this implies that there is a unique $\Sigma\Pi\Sigma(2)$ circuit for $\text{sim}(f)|_\rho$ of degree larger than D_4 , if such a circuit exists. Combining the two lemmas we see that there is a set S and an assignment ρ for which there is a unique $\Sigma\Pi\Sigma(2)$ circuit of degree larger than D_4 for $\text{sim}(f)|_\rho$ (the set S is picked so that the degree of the circuit for $\text{sim}(f)_\rho$ is larger than D_4 w.h.p. over the choice of ρ). At this point we will have at hand the unique circuit for $\text{sim}(f)|_\rho$ and we would like to add to it the variables not in S to get the circuit for $\text{sim}(f)$. To do so we find (again the unique) $\Sigma\Pi\Sigma(2)$ circuit for $\text{sim}(f)|_{\rho_{i,j}}$. From that circuit we can immediately find whether x_i and x_j belong to the same linear function (for each of the multiplication gates), and what is the ratio between their coefficients. Given this information for all pairs of variables it is easy to construct the two multiplication gates. We now give the formal proof.

Let $S \subset [n]$ be a set of size $2D_4$ and $C = M_1 + M_2$ be a multilinear $\Sigma\Pi\Sigma(2)$ circuit for $\text{sim}(f)$ of degree higher than D_4 . We say that the assignment ρ is *good* for C if no new linear factors were introduced (note that the degree of $\text{sim}(f)|_\rho$ may be smaller though). In other words, ρ is good if $\text{Lin}(\text{sim}(f)|_\rho) = 1$.

Lemma 14. *Let $S \subset [n]$ be a set of size $2D_4$ and $C = M_1 + M_2$ be a multilinear $\Sigma\Pi\Sigma(2)$ circuit for $\text{sim}(f)$ of degree $D > D_4$. The probability that an assignment ρ , to the variables not in S , is not good for C is smaller than $(n + 1)^2/|\mathbb{F}|$.*

The proof of the lemma is a simple union bound over the event that two linearly independent linear functions in the circuit for $\text{sim}(f)$ become linearly dependent non constant linear functions after substituting ρ .

Proof. We first bound the probability that M_1 or M_2 were set to zero. This may only happen if a linear function from any of them was set to zero and this happens with probability $1/|\mathbb{F}|$. As there are $2D$ linear functions we get an upper bound on the probability of $2D/|\mathbb{F}|$ (recall that we only work with the variables that do not belong to $\text{Lin}(f)$). To get a bound on the probability that a new linear factor was introduced, we note that this is possible only if there is a linear function L_1 dividing M_1 and a linear function L_2 dividing M_2 such that $L_1|_\rho \sim L_2|_\rho$, and they were not set to constants. As there is only one possible coefficient c for which $L_1|_\rho = c \cdot L_2|_\rho$ (i.e. c only depends on S, L_1 and L_2 and not on ρ), we see that the probability that this happened is equal to the probability that the part in $L_1 - cL_2$ that depends on the variables outside S was set to zero. As before this happens with probability $1/|\mathbb{F}|$. As there are at most D^2 such pairs we get an upper bound of $D^2/|\mathbb{F}|$ on the probability. Thus, the overall probability is bounded by $(D^2 + 2D)/|\mathbb{F}| < (D + 1)^2/|\mathbb{F}| \leq (n + 1)^2/|\mathbb{F}|$. \square

The next lemma shows that if there is a multilinear $\Sigma\Pi\Sigma(2)$ circuit $C = M_1 + M_2$, of degree $D > D_4$, for $\text{sim}(f)$ then it is unique and that if S and ρ are such that ρ is good for C then there is a unique multilinear $\Sigma\Pi\Sigma(2)$ circuit for $\text{sim}(f)|_\rho$ (and that circuit is $M_1|_\rho + M_2|_\rho$). This lemma generalizes Corollary 7 (by taking $S = [n]$ and ρ to be the “empty” substitution we get the claim regarding the uniqueness of the circuit for $\text{sim}(f)$).

Lemma 15. *Let $\text{sim}(f)$ be computed by a multilinear $\Sigma\Pi\Sigma(2)$ circuit $C = M_1 + M_2$. For a set S and a good assignment ρ , if $\deg(\text{sim}(f)|_\rho) > D_4$ then there is a unique simple multilinear $\Sigma\Pi\Sigma(2)$ circuit that computes $\text{sim}(f)|_\rho$. Moreover, this circuit is $M_1|_\rho + M_2|_\rho$.*

Proof. By our assumption we have that $\deg(\text{sim}(f)|_\rho) > D_4$. Assume for a contradiction that there is a multilinear $\Sigma\Pi\Sigma(2)$ circuit $C' = A_1 + A_2$ such that $C' = \text{sim}(f)|_\rho$ and $C' \neq M_1|_\rho + M_2|_\rho$. Therefore the circuit $\tilde{C} = M_1|_\rho + M_2|_\rho - A_1 - A_2$ is identically zero, of degree $> D_4$. Notice that as ρ is a good assignment we have that \tilde{C} is simple. By Corollary 4 we are assured that it is not minimal and so we must have that $A_1 = M_1|_\rho$ or $A_1 = M_2|_\rho$, which is a contradiction. \square

Given the two lemmas it is easy to explain Steps 1 and 2 of the algorithm. Let C be the unique multilinear $\Sigma\Pi\Sigma(2)$ circuit for $\text{sim}(f)$ of degree higher than D_4 . Let S be a set of size $2D_4$ such that the degree of C in the variables of S is larger than D_4 (think of the other variables as constants and view C as a circuit over the variables in S and compute its degree). By Lemma 14 we get that with probability $\geq 1 - (n + 1)^2/|F|$ the substitution ρ is such that $C|_\rho$ is the unique circuit for $\text{sim}(f)|_\rho$. Thus, for this set S we get that with probability $\geq 1 - (n + 1)^2/|F|$ Steps 1 and 2 find the unique circuit $C|_\rho$ (and this circuit is of degree greater than D_4). We now continue and show that if such S and good ρ were found then Steps 3 and 4 return the circuit C .

As ρ is good for C it immediately follows that $\rho_{i,j}$ is good for C (and $S \cup \{i, j\}$). Thus the circuits found in Step 3 are the unique circuits computing the different $f|_{\rho_{i,j}}$ -s (i.e. those circuits are identical to the $C|_{\rho_{i,j}}$ -s). We now explain how Step 4 is performed. Consider a linear function L that belongs to M_1 . There are two possibilities. Either $L|_\rho$ is constant

or not. If $L|_\rho$ is not constant then for every variable x_i we will find its coefficient in L by considering the circuit, say, $C|_{\rho_{1,i}}$ (assuming that $i \neq 1$, otherwise consider the circuit $C|_{\rho_{1,2}}$). If L was restricted to a constant, then denote $L = \sum_{i=1}^n \alpha_i \cdot x_i$. Notice that in $C|_{\rho_{i,j}}$ there will be a linear function of the form $c \cdot (\alpha_i \cdot x_i + \alpha_j \cdot x_j)$, for some non-zero field element c . In particular, if for some chosen x_i that appears in L , we chose to normalize its coefficient to always be 1 then we can find in this manner all the coefficient of the other variables appearing with it in L , and so reconstruct (a non-zero multiple of) L . In this way we can find all the linear functions appearing in M_1 and M_2 . Note that we found all the linear factors up to some non-zero multiple so it only remains to find a constant that will give the correct normalization of each M_i . This can be easily done by querying $\text{sim}(f)$ on two points, one on which M_1 vanishes and not M_2 and viceversa. We note that we do not have to verify that we found the correct circuit for $\text{sim}(f)$ as by Lemma 15 and Step 2 we are guaranteed that we have the correct circuit for $\text{sim}(f)|_\rho$ (assuming that ρ is good) and it is clear that starting from this circuit we find the unique circuit C for $\text{sim}(f)$. Thus, with probability at least $1 - (n + 1)^2/|F|$ the algorithm finds the unique multilinear circuit for $\text{sim}(f)$.

It only remain to analyze the running time of the algorithm. Given a set S and an assignment ρ we check whether $\text{sim}(f)|_\rho$ has new linear factors by applying Theorem 8, which requires polynomial time and succeeds w.h.p. In Step 2 we have to go over all possible simple multilinear $\Sigma\Pi\Sigma(2)$ circuits for $\text{sim}(f)|_\rho$ of degree greater than D_4 (and at most $2D_4$). Note that as there are only $|\mathbb{F}|^{2D_4+1}$ linear functions in the variables $\{x_i\}_{i \in S}$ then there are polynomially many such circuits ($|\mathbb{F}|^{2D_4 \cdot (2D_4+1)}$ is an obvious upper bound). Among all those circuits we shall find, by going over all 0, 1 assignments to $\{x_i\}_{i \in S}$, a circuit that computes $\text{sim}(f)|_\rho$. As the size of S is constant this only costs a constant factor in the running time. Verifying that the a circuit is simple is pretty obvious and requires time polynomial in the degree of the circuit and linear in n . Next, for each pair (i, j) the complexity of Step 3 is similar to the complexity of Step 2. Finally, the procedure described above for “gluing” the different $C|_{\rho_{i,j}}$ -s require polynomial time (in n). Hence, the over all running time is polynomial in n and $|\mathbb{F}|$ and with probability $\geq 1 - (n + 1)^2/|F|$ the algorithm outputs the unique multilinear $\Sigma\Pi\Sigma(2)$ circuit for $\text{sim}(f)$. Note that failure can happen only if we failed to find a good ρ . This completes the proof of Theorem 13. \square

We now combine the preprocessing step and Theorems 10 and 13 to get Theorem 1 (the argument is straight forward).

3.4 Proof of Theorem 1

The proof of the Theorem is easy given the two algorithms. First we find $\text{Lin}(f)$ and compute $D = d - \deg(\text{Lin}(f))$. If $D \leq D_4$ then we run the low degree algorithm. If $D > D_4$ then we run the high degree algorithm. We are assured that if we get an answer then it is going to be correct. The probability of failure, in both algorithms is polynomially small in n (and can be easily decreased, e.g. by repeating the algorithm) and so the theorem follows.

4 General Circuits

In this section we prove Theorem 2. For convenience we repeat it here.

Theorem 2 Let f be an n -variate polynomial computed by a $\Sigma\Pi\Sigma(2)$ circuit of degree d , over a field \mathbb{F} . Then there is a randomized interpolation algorithm that given black box access to f and the parameters d and n runs in quasi-polynomial time (in $n, d, |\mathbb{F}|$) and has the following properties:

- If $\text{rank}(f) = \Omega(\log^2(d))$, then with probability $1 - o(1)$ the algorithm outputs the (unique) $\Sigma\Pi\Sigma(2)$ circuit for f .
- If $\text{rank}(f) = O(\log^2(d))$, then the algorithm outputs, with probability $1 - o(1)$, a polynomial $\text{Lin}(f)$, a polynomial $Q(y_1, \dots, y_k)$ and k linear functions L_1, \dots, L_k , where $k \leq \text{rank}(f)$, such that $\text{Lin}(f)$ is a the product of all the linear factors of f and $\text{Lin}(f) \cdot Q(L_1, \dots, L_k) = f$.

When $|\mathbb{F}| < \max(d^5, n^5)$ the algorithm is allowed to make queries to f from a polynomial size extension field of \mathbb{F} .

From now on we shall assume, w.l.o.g., that the underlying field \mathbb{F} is of size greater than $\max(d^5, n^5)$. We shall have the following representation of f in mind:

$$f(\bar{x}) = \prod_{i=1}^d L_i^{(1)}(\bar{x}) + \prod_{i=1}^d L_i^{(2)}(\bar{x}). \quad (8)$$

Note that in order to ease the notations we assume that the degrees of both multiplication gates are equal. As described in Section 1.4 there are two conceptual steps in the proof. First we restrict the inputs to come from a random subspace V of dimension $O(\log^2(d))$, where $d = \deg(f)$. We then learn the restriction of f to V which we denote with $f|_V$. The second step is to learn the restriction of f to larger subspaces and then glue the different representations together. While this is the general picture there is a difference in the way that we handle functions with high rank and functions with low rank. We start by describing the algorithm for the low rank case

4.1 Interpolation for the low rank case

In this section we give an interpolation algorithm for polynomials computed by $\Sigma\Pi\Sigma(2)$ circuits of degree d , that have rank at most $10R_4 \cdot \log^2(d)$ (recall the definition of R_4 from Corollary 6). Let f be such a polynomial. Algorithm 3 is an interpolation algorithm for f . Before giving the algorithm we describe each of its steps.

Let C be any $\Sigma\Pi\Sigma(2)$ circuit computing f , e.g. the one given by Equation 8. We start by computing $\text{Lin}(f)$. This can be easily done using Theorem 8. After this step we can assume, as before, that we get oracle access to¹¹ $\text{sim}(f) = f/\text{Lin}(f)$. Note, that as $\text{sim}(f) = f/\text{Lin}(f)$ divides $\text{sim}(C)$ and $\text{rank}(f) \leq 10R_4 \cdot \log^2(d)$, we get that $\text{sim}(f)$ can be represented as a polynomial in $\leq 10R_4 \cdot \log^2(d)$ linear functions. Namely, there exists a polynomial $P(y_1, \dots, y_k)$ and linearly independent linear functions $\{L_i\}_{i \in [k]}$ such that $\text{sim}(f) = P(L_1, \dots, L_k)$ (we will always think of k as the minimal possible integer allowing such a representation). The goal of Algorithm 3 is to find such a representation for $\text{sim}(f)$. There are two main steps for doing that. First we pick a random subspace V of dimension $O(R_4 \cdot \log^2(d))$ and look for a polynomial Q and linear functions ℓ_1, \dots, ℓ_k such that $\text{sim}(f)|_V = Q(\ell_1, \dots, \ell_k)$ (we do not necessarily find the polynomial P and restricted linear functions $\{L_i|_V\}$ given above, however the two different representations are closely related as shown in Lemma 23). In the second step we “lift” the ℓ_i -s and find ℓ_i^* -s such that for every i , $\ell_i^*|_V = \ell_i$ and $\text{sim}(f) = Q(\ell_1^*, \dots, \ell_k^*)$. This may seem as a big leap from the outcome of the first step, however we manage to do so by exposing one variable at a time (in a similar way to Step 4 of Algorithm 1). The important ingredients of this “lifting” procedure are Lemmas 23 and 24 that basically show that if V' is a subspace containing V then there exist unique linear functions (that can be easily found) $\{\ell'_i\}_{i=1}^k$, over V' , that satisfy $\text{span}(\{L_i|_{V'}\}) = \text{span}(\{\ell'_i\})$, $\ell'_i|_V = \ell_i$ and $Q(\ell'_1, \dots, \ell'_k) = \text{sim}(f)|_{V'}$. We now give the formal description of the algorithm.

The next theorem shows the correctness of Algorithm 3 and analyzes its running time.

Theorem 16. *Let f be a polynomial that can be computed by a $\Sigma\Pi\Sigma(2)$ circuit of rank $\leq 10R_4 \cdot \log^2(d)$. Then with probability $\geq 1 - |\mathbb{F}|^{-\Omega(\log^2(n))}$ Algorithm 3, when given oracle access to f , computes $\text{Lin}(f)$, a natural number $k \leq 10R_4 \cdot \log^2(d)$, a polynomial $Q(y_1, \dots, y_k)$ and k linear functions $\{\ell_i^*\}_{i=1 \dots k}$ such that $Q(\ell_1^*, \dots, \ell_k^*) = \text{sim}(f)$. The running time of the algorithm is quasi-polynomial in n, d and $|\mathbb{F}|$ (i.e. it is $\exp(\text{poly}(\log |\mathbb{F}|, \log n, \log d))$).*

The proof of correctness of the algorithm is composed of three parts. First we study properties of polynomials that can be written as functions of exactly k linear functions. In particular we show that this property is preserved (w.h.p.) when we restrict the polynomial to a random subspace of sufficient dimension (Corollary 22). Then we consider $\text{sim}(f)$ and a random subspace V of dimension $10R_4 \cdot \log^2(d)$ and show how to find the relevant linear functions for it. This will complete the analysis of Step 2. After that we prove a lemma mentioned above (Lemmas 24) that shows how we can “lift” the linear functions found in the previous step to functions over \mathbb{F}^n . This will complete the analysis of Step 3. The analysis of the running time will be quite simple. For sake of modularity we discuss properties of polynomials that depend on k linear functions in the next subsection and then proceed with the proof.

¹¹Unlike the multilinear case it may be the case that $\text{sim}(C)$ does not compute $\text{sim}(f)$.

Algorithm 3 - General $\Sigma\Pi\Sigma(2)$ circuits of low rank

1. **Computing** $\text{Lin}(f)$: Find the linear factors of f using the factoring algorithm of Theorem 8.
2. **Interpolating on a low dimensional random subspace**: Pick a random subspace V of dimension $s = 20R_4 \cdot \log^2(d) + \log^2(n)$. Find $k' \leq 10R_4 \cdot \log^2(d)$ linearly independent linear functions $\{\ell_i\}_{i \in [k']}$, such that $\text{sim}(f)|_V = Q(\ell_1, \dots, \ell_{k'})$, for some polynomial Q of degree $\deg(Q) \leq d$, and such that no such representation is possible for any $k'' < k'$. In other words do the following: For $k' = 1 \dots 10R_4 \cdot \log^2(d)$ consider all sets of k' linearly independent linear functions over V . For every such set $\{\ell_i\}_{i \in [k']}$ find, using brute force interpolation, a polynomial $Q(y_1, \dots, y_{k'})$ of degree at most d , such that $Q(\ell_1, \dots, \ell_{k'}) = \text{sim}(f)|_V$, if such a polynomial exists. Finally output the first set $\{\ell_i\}_{i \in [k']}$ for which such a polynomial Q was found.
3. **Lifting from V to \mathbb{F}^n** : Find linear functions $\{\ell_i^*\}_{i \in [k']}$, over \mathbb{F}^n , such that $Q(\ell_1^*, \dots, \ell_{k'}^*) = \text{sim}(f)$ using the following procedure (a more detailed explanation will be given when analyzing the algorithm):
 - (a) Let $\{v_i\}_{i \in [n]}$ be a basis to \mathbb{F}^n , such that $\{v_i\}_{i=1}^s$ is a basis to V . For every $i \in [n-s]$ let $V_i = \text{span}(V \cup \{v_{s+i}\})$. For each V_i find a representation for $\text{sim}(f)|_{V_i}$ of the form $\text{sim}(f)|_{V_i} = Q(\ell_1^{(i)}, \dots, \ell_k^{(i)})$, where for each i and j we have that $\ell_j^{(i)}|_V = \ell_j$.
 - (b) Find the unique linear functions $\{\ell_i^*\}_{i \in [k]}$ that satisfy $\ell_j^*|_{V_i} = \ell_j^{(i)}$, for each i and j , in an analogous way to Step 4 of Algorithm 1.

If no such representation is found then output “fail”.

4.1.1 Polynomials depending on k linear functions

In this subsection we study the notion of a polynomial that depends on exactly k linear functions. We first define this property in a formal manner.

Definition 17. Let $h(x_1, \dots, x_n)$ be a polynomial. We say that h is a polynomial in **exactly** k linear functions if there is a polynomial $P(y_1, \dots, y_k)$ and k linear functions L_1, \dots, L_k such that $h = P(L_1, \dots, L_k)$, and there is no such representation for h with less than k linear functions. We say that h is a polynomial in k linear functions if there exists a $k' \leq k$ such that h is a polynomial in exactly k' linear functions.

The following lemma gives a sufficient and necessary condition for being a function of k linear functions.

Lemma 18. Let $h(x_1, \dots, x_n)$ be a polynomial over \mathbb{F} . Then h can be written as a polynomial in k linear functions if and only if there is a subspace $V^* \subseteq \mathbb{F}^n$ of dimension $n - k$ such that for every $u \in \mathbb{F}^n$ and $v \in V^*$ we have that $h(u) = h(v + u)$.

Note that the lemma does not speak about h being a function of exactly k linear functions. However, it is clear that if there is a subspace V^* of dimension $n-k$ that satisfies the condition in the lemma, and there is no subspace U^* of dimension $> n-k$ that satisfies the condition of the lemma, then h is a function of exactly k linear functions.

Proof. Assume w.l.o.g. that h is a polynomial in exactly k linear functions. Denote $h = P(L_1, \dots, L_k)$, where the L_i -s are homogeneous linear functions. Let $V^* = \{\bar{x} \in \mathbb{F}^n : \forall i L_i(\bar{x}) = 0\}$. Clearly $\dim(V^*) = n-k$. Let $u \in \mathbb{F}^n$ and $v \in V^*$ be two vectors. We have that $h(v+u) = P(L_1(u+v), \dots, L_k(v+u)) = P(L_1(u), \dots, L_k(u)) = h(u)$.

For the other direction, given such V^* let L_1, \dots, L_k be k linearly independent linear functions such that for every $v \in V^*$ and $i \in [k]$ we have that $L_i(v) = 0$. Let $V \subset \mathbb{F}^n$ be a k -dimensional subspace such that $V \cap V^* = \{0\}$. Clearly $\mathbb{F}^n = V \oplus V^*$. It is also clear that the linear functions $\{L_i|_V\}$ are linearly independent. As $\dim(V) = k$ it follows that there is some polynomial $Q(y_1, \dots, y_k)$ such that for every $v \in V$ it holds that $h(v) = Q(L_1(v), \dots, L_k(v))$. Now, given $u \in \mathbb{F}^n$ write $u = v + v^*$ for $v \in V$ and $v^* \in V^*$. We now have that $h(u) = h(u - v^*) = h(v) = Q(L_1(v), \dots, L_k(v)) = Q(L_1(v + v^*), \dots, L_k(v + v^*)) = Q(L_1(u), \dots, L_k(u))$. Hence, $h = Q(L_1, \dots, L_k)$. \square

We now give some easy corollaries of this lemma.

Corollary 19. *Let $h(\bar{x}) = P(L_1, \dots, L_k)$ be a polynomial in exactly k linear functions. Let V be a subspace of \mathbb{F}^n . Then $h|_V$ is a polynomial in exactly k linear functions if and only if the restrictions $L_1|_V, \dots, L_k|_V$ are linearly independent.*

Corollary 20. *Let h be a polynomial in exactly k linear functions. Let $V \subseteq \mathbb{F}^n$ be an s -dimensional subspace such that $h|_V$ is a polynomial in $k' < k$ linear functions. Let v_1, \dots, v_{n-s} be vectors such that $\text{span}(V \cup \{v_i\}_{i \in [n-s]}) = \mathbb{F}^n$. Denote $V_i = \text{span}(V \cup \{v_i\})$. Then for some $i \in [n-s]$ we have that $h|_{V_i}$ is a polynomial in more than k' linear functions.*

In the analysis of Algorithm 3 we will be interested in restrictions to random subspaces. The following lemmas show that if h is a polynomial in exactly k linear functions and V is a subspace of sufficiently high dimension chosen at random then w.h.p. $h|_V$ is also a polynomial in exactly k linear functions.

Lemma 21. *Let $\{\ell_i\}_{i \in [t]}$ be a set of linearly independent linear functions over \mathbb{F}^n . Let $V \subseteq \mathbb{F}^n$ be a random s -dimensional subspace. Then the probability that the set $\{\ell_i|_V\}_{i \in [t]}$ is linearly dependent is at most $|\mathbb{F}|^{t-s}$.*

Proof. Clearly, ℓ_1, \dots, ℓ_t are linearly dependent on V if and only if there is a nonzero linear combination $\alpha_1 \ell_1 + \dots + \alpha_t \ell_t$ that vanishes on V . Given V let us define $\mathcal{L}^* = \{\ell(x_1, \dots, x_n) : \ell|_V = 0\}$. Note that as V is random then so is \mathcal{L}^* . Namely, \mathcal{L}^* is a random $n-s$ -dimensional subspace of linear functions. Thus, ℓ_1, \dots, ℓ_t are linearly dependent on V if and only if there is a nonzero function in the span of the ℓ_i -s that belongs to \mathcal{L}^* . In other words, we have to bound the probability that a random subspace of dimension $n-s$ (i.e. \mathcal{L}^*) intersects a given subspace of dimension $\leq t$ (i.e. $\text{span}(\ell_1, \dots, \ell_t)$). As the probability that a random nonzero vector belongs to a given t dimensional subspace is $(|\mathbb{F}|^t - 1)/(|\mathbb{F}|^n - 1)$ we get

by the union bound that the probability of a non trivial intersection is upper bounded by $(|\mathbb{F}|^{n-s} - 1) \cdot (|\mathbb{F}|^t - 1) / (|\mathbb{F}^n| - 1) < |\mathbb{F}|^{t-s}$. \square

From Corollary 19 and Lemma 21 we get the following corollary.

Corollary 22. *Let h be a function of exactly k linear forms¹². Let V be a random subspace of dimension $\geq k + \log^2(n)$. Then the probability that $h|_V$ is not a function of exactly k linear forms is at most $|\mathbb{F}|^{-\log^2(n)}$.*

The next lemma shows that if a polynomial in k linear functions, h , has two different representations as a polynomial in exactly k linear functions then these representations are closely related.

Lemma 23. *Let $h(\bar{x})$ be a polynomial in exactly k linear functions. Let $P(\ell'_1, \dots, \ell'_k) = h = Q(\ell_1, \dots, \ell_k)$ be two different representations for h . Then $\text{span}(\{\ell'_i\}_{i \in [k]}) = \text{span}(\{\ell_i\}_{i \in [k]})$. In particular this implies that there is an invertible linear transformation $T : \mathbb{F}^k \rightarrow \mathbb{F}^k$ such that $Q(y_1, \dots, y_k) = (P \circ T)(y_1, \dots, y_k)$.*

Proof. Assume for a contradiction that, w.l.o.g., $\ell'_1, \ell_1, \dots, \ell_k$ are linearly independent. Let W be the co-dimension 1 subspace on which ℓ'_1 vanishes. It is clear that $\ell_1|_W, \dots, \ell_k|_W$ are linearly independent, whereas $\ell'_1|_W, \dots, \ell'_k|_W$ are linearly dependent (as $\ell'_1|_W = 0$). In particular by Corollary 19 we get that $h|_W$ is a function of at most $k - 1$ linear functions (when considering the representation according to the $\ell'_i|_W$ -s), and a function of exactly k linear functions (when considering the representation according to the $\ell_i|_W$ -s), which is a contradiction. Now, let $T : \mathbb{F}^n \rightarrow \mathbb{F}^n$ be an invertible linear transformation taking $(\ell_i)_i$ to $(\ell'_i)_i$. Namely, $T(\ell_1(v), \dots, \ell_k(v)) = (\ell'_1(v), \dots, \ell'_k(v))$ for every $v \in \mathbb{F}^n$. We get that for every $v \in \mathbb{F}^n$ it holds that

$$P \circ T(\ell_1(v), \dots, \ell_k(v)) = P(\ell'_1(v), \dots, \ell'_k(v)) = h(v) = Q(\ell_1(v), \dots, \ell_k(v)).$$

As the ℓ_i -s are linearly independent it follows that we must have $P \circ T = Q$ (as otherwise we will have two low degree polynomials representing the same function over a large field). \square

4.1.2 Proof of Theorem 16

We are now ready to give the proof of Theorem 16.

Proof of Theorem 16. By our assumption that $\text{rank}(f) \leq 10R_4 \cdot \log^2(d)$ we get that $\text{sim}(f)$ can be written as a polynomial in at most $10R_4 \cdot \log^2(d)$ linear functions. For simplicity we shall assume that it is a polynomial of exactly $k = \text{rank}(f)$ linear functions. Let

$$\text{sim}(f) = P(L_1, \dots, L_k) \tag{9}$$

where $P(y_1, \dots, y_k)$ is a polynomial and $\{L_i\}_{i=1}^k$ are linear forms. We shall later use this representation. We now analyze separately each step of the algorithm. Step 1 is an immediate application of Theorem 8 and does not require special analysis.

¹²A linear form is a homogeneous linear function.

Step 2: Interpolating on a low dimensional random subspace

Recall that in this step we pick a random subspace $V \subseteq \mathbb{F}^n$ of dimension $s = 20R_4 \cdot \log^2(d) + \log^2(n)$, and interpolate $\text{sim}(f)$ over it. By Corollary 22 we get that with probability $\geq 1 - |F|^{-\log^2(n)}$ the polynomial $\text{sim}(f)|_V$ is a polynomial in exactly k linear functions. We now find a representation of the form $\text{sim}(f)|_V = Q(\ell_1, \dots, \ell_k)$.

The idea is pretty simple. We go over all possible values for k (recall that we do not know k in advance). For each $k' \leq 10R_4 \cdot \log^2(d) + \log^2(n)$ we go over all possible sets of k' linearly independent linear functions over V , and for each set $\{\ell_i\}_{i \in [k']}$ try to represent $\text{sim}(f)|_V$ as a polynomial $Q(\ell_1, \dots, \ell_{k'})$. Note that given $\{\ell_i\}_{i \in [k']}$ we can find such a polynomial Q (if such Q exists) by a simple brute-force interpolation. As this is routine we skip it here (for completeness we give a description of this process in Appendix A).

Using the approach above we may get many different representations for $\text{sim}(f)|_V$. Among all those representations we pick one with the smallest possible k' . As mentioned above, by Corollary 22 we get that w.h.p. $k' = k$. Moreover, Corollary 20 gives a way for verifying that indeed $k' = k$. So, to conclude, after the first step of the algorithm we can be sure that¹³ $k' = k$ and that we have a representation of the form $\text{sim}(f)|_V = Q(\ell_1, \dots, \ell_k)$.

The running time of this step is dominated by the number of different subsets of at most $10R_4 \cdot \log^2(d)$ linearly independent linear functions over V . Clearly there are at most $|\mathbb{F}|^{10R_4 \cdot \log^2(d) + \log^2(n)}$ linear functions over V and so we go over at most $|\mathbb{F}|^{(10R_4 \cdot \log^2(d) + \log^2(n)) \cdot (10R_4 \cdot \log^2(d))}$ such subsets. The brute force interpolation algorithm (as described in Appendix A) is slightly faster and so the total running time of this step is $|\mathbb{F}|^{O(\log^2(d) \cdot (\log^2(d) + \log^2(n)))} = \exp(\log |\mathbb{F}| \cdot \log^2(d) \cdot (\log^2(d) + \log^2(n)))$.

We now move to the next step in which we show how to lift the representation of $\text{sim}(f)|_V$ to \mathbb{F}^n .

Step 3: Lifting from V to \mathbb{F}^n

Recall that in this step we look for linear functions $\{\ell_i^*\}_{i \in [k]}$ such that $\ell_i^*|_V = \ell_i$ and $\text{sim}(f) = Q(\ell_1^*, \dots, \ell_k^*)$. The existence of such functions will follow from the next lemma.

Lemma 24. *Let $\{L_i\}_{i \in [k]}$ be the linear functions in Equation (9) and $\{\ell_i\}_{i \in [k]}$ be the linear functions found in Step 2 of Algorithm 3. Assume that $\{\ell_i^*\}_{i \in [k]}$ satisfy $\ell_i^*|_V = \ell_i$ and $\text{span}(\{\ell_j^*\}_{j \in [k]}) = \text{span}(\{L_j\}_{j \in [k]})$. Then $Q(\ell_1^*, \dots, \ell_k^*) = \text{sim}(f)$.*

Proof. We shall use the same notations as in Equation (9). By the assumption that $\text{span}(\{\ell_j^*\}_{j \in [k]}) = \text{span}(\{L_j\}_{j \in [k]})$ we see that there is a polynomial $Q'(y_1, \dots, y_k)$, of degree $\deg(Q') = \deg(P)$, such that $Q'(\ell_1^*, \dots, \ell_k^*) = P(L_1, \dots, L_k) = \text{sim}(f)$. By considering the restriction to V we get

$$Q(\ell_1, \dots, \ell_k) = \text{sim}(f)|_V = P(L_1, \dots, L_k)|_V = Q'(\ell_1^*, \dots, \ell_k^*)|_V$$

¹³In the algorithm we do not check whether $k' = k$ (by going one dimension up, using Corollary 20) but we can just as well add such a check without much change to the running time. For the sake of simplicity we did not add this step to the algorithm.

$$= Q'(\ell_1^*|_V, \dots, \ell_k^*|_V) = Q'(\ell_1, \dots, \ell_k).$$

As the functions $\{\ell_i\}_{i \in [k]}$ are linearly independent, and the degrees of Q and Q' are smaller than the size of the field, we get that $Q \equiv Q'$. In other words, $Q(y_1, \dots, y_k)$ and $Q'(y_1, \dots, y_k)$ are the same polynomial. In particular $Q(\ell_1^*, \dots, \ell_k^*) = Q'(\ell_1^*, \dots, \ell_k^*) = \text{sim}(f)$, which is what we wanted to prove. \square

With the help of the lemma we now explain why Step 3 indeed finds the required representation for $\text{sim}(f)$. In that step we first construct $n - s$ linear spaces $\{V_i\}_{i \in [n-s]}$ of dimension $s + 1$, such that $\cap_i V_i = V$ and $\text{span}(\cup_i V_i) = \mathbb{F}^n$. For each such subspace we look for linear functions $\{\ell_j^{(i)}\}_{j \in [k]}$ such that $\ell_j^{(i)}|_V = \ell_j$, for every $j \in [k]$, and $Q(\ell_1^{(i)}, \dots, \ell_k^{(i)}) = \text{sim}(f)|_{V_i}$. The reason that such $\ell_j^{(i)}$ -s exist follows from Lemmas 23 and 24. Indeed, Lemma 23 implies that $\text{span}(\{L_i|_V\}_{i \in [k]}) = \text{span}(\{\ell_i\}_{i \in [k]})$. Hence, if we consider the $k \times n$ matrix A , whose rows correspond to the L_t -s and columns correspond to the basis vectors $\{v_i\}_{i \in [n]}$, in which the (t, j) -th position equals $L_t(v_j)$, then there exists an invertible linear transformation $T : \mathbb{F}^k \rightarrow \mathbb{F}^k$, such that for $1 \leq j \leq k$, the (t, j) -th entry in $T \cdot A$ is equal to $\ell_t(v_j)$. It follows that if we consider the linear functions $\ell_t^{(i)}$ defined by $\ell_t^{(i)}(v_j) = (T \cdot A)_{t,j}$, for $j \in [k] \cup \{i\}$, then this gives the required $\ell_t^{(i)}$ -s. In particular such $\ell_t^{(i)}$ -s exist and we can find them by a simple interpolation. Therefore we are guaranteed that Step 3a succeeds. Moreover, this argument also shows the uniqueness of the $\ell_j^{(i)}$ -s (this follows from the fact that T is invertible). Hence, by the explanation above it is clear that we can easily construct the matrix $T \cdot A$ by simply letting $(T \cdot A)_{t,j} = \ell_t^{(j)}(v_j)$ for $k < j$, and $(T \cdot A)_{t,j} = \ell_t(v_j)$ for $j \leq k$. If we now pick ℓ_i^* to be the linear function defined by $\ell_i^*(v_j) = (T \cdot A)_{i,j}$, for $j \in [n]$, then by Lemma 24 we are guaranteed that $Q(\ell_1^*, \dots, \ell_k^*) = \text{sim}(f)$. This shows the correctness of Step 3b (and the algorithmic way for constructing the ℓ_i^* -s). An interesting thing to notice is that we have no knowledge of the linear functions $\{L_i\}$ nor of the linear transformation T , yet we are able to construct the matrix $T \cdot A$ using the (simple) conclusions of Lemmas 23 and 24.

The running time of Step 3 is at most a polynomial factor times the running time of Step 2. The main factor is repeating Step 2 for each V_i . Then, finding the ℓ_i^* -s is quite simple and can be done in time polynomial in n , as described above. This completes the proof of Theorem 16. \square

We note that as $k = O(\log^2(d))$ and $\deg(Q) \leq d$ then $Q(\ell_1^*, \dots, \ell_k^*)$ can be represented as a depth-3 circuit with quasi-polynomially many multiplication gates.

4.2 Interpolation for the high rank case

In this section we deal with the case that f has a circuit of rank higher than $10R_4 \cdot \log^2(d)$. We shall use the notation $f = M_1 + M_2$, where M_1 and M_2 are the two multiplication gates in the $\Sigma\Pi\Sigma(2)$ circuit for f . We note that from Corollary 7 and the assumption that $\text{rank}(f) \geq 10R_4 \cdot \log^2(d)$ we get that f has a unique $\Sigma\Pi\Sigma(2)$ circuit of degree d .

Before giving the algorithm itself we first describe its main steps. The algorithm follows the sketch of Section 1.4. We first restrict the circuit to a low dimensional space V (where low means $\text{poly}(\log d)$). Then we reconstruct $f|_V$, which is the main technical difficulty of the algorithm. Finally we lift the circuit for $f|_V$ to a circuit for f . This step is done in a similar manner to the lifting process of Algorithm 2 (Steps 3 and 4 there), and is based on the uniqueness of the circuit for f . We now say a few more words on the way to find a circuit for $f|_V$. The idea is to find a large set (of size $O(\log^2(d))$) of linearly independent linear functions $\{\ell_i\}$ that all divide, say, $M_1/\text{gcd}(C)$. Such a set exists because of the assumption on the rank of f . Then we consider each of those linear functions and define the subspace $V_i = V|_{\ell_i=0}$ (i.e. the subspace of V on which ℓ_i vanishes¹⁴). Note that for every i it holds that $M_1|_{V_i} = 0$ and so by querying f on V_i we get oracle access to $M_2|_{V_i}$. Thus, by simple factoring we get all the circuits $M_2|_{V_i}$. The challenge now is to combine all of them to get the circuit $M_2|_V$. We later discuss this step in more detail. We now give the formal algorithm (Algorithm 4).

For convenience we divide the analysis of the three steps of the algorithm to three different subsections. We begin with the analysis of Step 1.

4.2.1 Step 1: Interpolating on a low dimensional subspace I

Lemma 25. *With probability $\geq 1 - |\mathbb{F}|^{-\Omega(\log^2 n)}$, over the choice of V , Step 1 of Algorithm 4, when given oracle access to f , finds¹⁵ a set of $t = 100 \log d$ linearly independent linear functions $\{\ell_i\}_{i \in [t]}$ such that $\prod_{i=1}^t \ell_i | M_j$ for some $j \in \{1, 2\}$, but no ℓ_i divides M_{3-j} (the other multiplication gate). The running time of Step 1 is $\exp(\text{poly}(\log n, \log d, \log |\mathbb{F}|))$.*

Proof. In the heart of the proof of the lemma is the fact that w.h.p., over the choice of V , we have that $\text{rank}(f)|_V \geq 10R_4 \cdot \log^2(d)$. We prove this fact in Lemma 26. It will be easy to show that when $\text{rank}(f|_V)$ is high then the required set $\{\ell_i\}$ exists. Recall that $\text{gcd}(C) = \text{g.c.d.}(M_1, M_2)$, and that

$$\text{sim}(C) = C / \text{gcd}(C) = M'_1 + M'_2 \tag{10}$$

is a $\Sigma\Pi\Sigma(2)$ circuit. From Corollary 7 and the assumption that $\text{rank}(f) \geq 10R_4 \cdot \log^2(d)$ we get that $\text{sim}(C)$ is the unique $\Sigma\Pi\Sigma(2)$ circuit for $f/\text{gcd}(C)$. Let $V \subseteq \mathbb{F}^n$ be a random subspace of dimension $s = 20R_4 \cdot \log^2(d) + \log^2(n)$. The following lemma shows that w.h.p. $\text{rank}(f|_V)$ is high (the intuition is the same as in Corollary 22).

Lemma 26. $\Pr[\text{rank}(f|_V) \leq 10R_4 \cdot \log^2(d)] \leq |\mathbb{F}|^{-\Omega(\log^2 n)}$.

Proof of Lemma 26. To prove the claim we show that w.h.p. the rank of the circuit $M_1|_V + M_2|_V$ is not too small, and then, by Corollary 7, we get that this is actually the *unique* representation of $f|_V$ as a $\Sigma\Pi\Sigma(2)$ circuit. The lemma will follow from this fact.

¹⁴This is actually an affine subspace, but for simplicity we assume that it is a linear space.

¹⁵Of course many other sets are found but only those sets interest us.

Algorithm 4 General circuits of high rank

1. **Interpolating on a low dimensional subspace I:** Pick a random subspace $V \subseteq \mathbb{F}^n$ of dimension $s \triangleq 20R_4 \cdot \log^2(d) + \log^2(n)$. Consider the restriction $f|_V$. For every set of $t = 100 \log(d)$ linearly independent linear functions $\{\ell_i\}_{i \in [t]}$ over V , check whether for every $i \in [t]$ the restriction of f to the (affine) subspace $V_i \triangleq V \cap \{x : \ell_i(x) = 0\}$ is equal to a product of linear functions (by factoring).
 2. **Interpolating on a low dimensional subspace II:** For each choice of $\{\ell_i\}_{i \in [t]}$, for which factoring was possible, merge (again, only if possible) the different factors into one multiplication gate (using Algorithm 5). For each multiplication gate found, M , check whether $f|_V - M$ is a product of linear functions (using the factoring algorithm from Theorem 8). If this is the case then output the representation found for $f|_V$. If no such representation is found then output “fail”.
 3. **Lifting from V to \mathbb{F}^n :** Lift the representation found over V to a representation over \mathbb{F}^n (using the same method as in Steps 3 and 4 of Algorithm 2). Namely, if $\{v_i\}_{i \in [s]}$ is a basis for V and $\{v_i\}_{i \in [n]}$ is a basis for \mathbb{F}^n that extends the basis of V , then let $W_i = \text{span}(V \cup v_i)$, for $s < i \leq n$. Repeat Steps 1 and 2 for each W_i to get the corresponding circuit C_i . If none of the steps failed for any of the W_i -s then combine the different C_i -s to get C . Combining the circuits is done as follows. Let $\{z_i\}_{i \in [n]}$ be the dual basis to $\{v_i\}_{i \in [n]}$, i.e. $z_i(v_j) = \delta_{i,j}$. Each linear function in $C \cup \{C_i\}_{i=s+1}^n$ can be written as a linear combination of the z_i -s and the constant function. Given a linear function ℓ in C , find the unique ℓ_i in C_i such that $\ell_i|_{z_i=0} = \ell$ (possibly after multiplying ℓ_i by a constant from \mathbb{F}). If no such ℓ_i exists or if it is not unique (i.e. there is $\ell'_i \not\sim \ell_i$ with $\ell'_i|_{z_i=0} = \ell$) then output “fail”. Otherwise, if $\ell = \ell_i + \alpha_i \cdot z_i$ then let $\ell' = \ell + \sum_{i=s+1}^n \alpha_i \cdot z_i$. Now, replace each linear function ℓ in C with the corresponding ℓ' . Denote the resulting circuit with C' . Output C' .
-

Lemma 27.

$$\Pr \left[\text{rank} \left(\frac{M_1|_V + M_2|_V}{\text{g.c.d.}(M_1|_V, M_2|_V)} \right) \leq 10R_4 \log^2(n) \right] \leq |\mathbb{F}|^{-\Omega(\log^2(n))}.$$

Proof of Lemma 27. In order to show that the rank of $M_1|_V + M_2|_V$ does not decrease by much, we first show that w.h.p. $\text{g.c.d.}(M_1|_V, M_2|_V) = \text{g.c.d.}(M_1, M_2)|_V$, and then using Lemma 21 we complete the proof.

Lemma 28. $\Pr[\text{g.c.d.}(M_1|_V, M_2|_V) \neq \text{g.c.d.}(M_1, M_2)|_V] \leq d^2(|\mathbb{F}| + 1)/|\mathbb{F}|^s = |\mathbb{F}|^{-\Omega(\log^2(n))}$.

Proof of Lemma 28. It is not hard to see that $\text{g.c.d.}(M_1|_V, M_2|_V)$ is larger than $\text{g.c.d.}(M_1, M_2)|_V$ only if there are two linearly independent linear functions L_1, L_2 , such that $L_i|_{M_i}$, and such that $L_1|_V \sim L_2|_V$. The probability, over the choice of V , that this will

happen is at most $(|\mathbb{F}| + 1)/|\mathbb{F}|^s$. As there are at most d^2 such pairs of linear function, the overall probability that the degree of the g.c.d. increases is at most $d^2(|\mathbb{F}| + 1)/|\mathbb{F}|^s$. \square

We continue with the proof of Lemma 27. We know that w.h.p. $\text{g.c.d.}(M_1|_V, M_2|_V) = \text{g.c.d.}(M_1, M_2)|_V$. Therefore, w.h.p.,

$$(M_1|_V + M_2|_V)/\text{g.c.d.}(M_1|_V, M_2|_V) = M'_1|_V + M'_2|_V$$

(as defined in Equation 10). By the assumption on $\text{rank}(f)$, we get that the rank of the linear functions in $M'_1 + M'_2$ is at least $10R_4 \cdot \log^2(d)$. The result now follows by using Lemma 21 with the parameters¹⁶ $s = 20R_4 \cdot \log^2(d) + \log^2(n)$ and $t = 10R_4 \cdot \log^2(d)$. The lemma implies that

$$\Pr [\text{rank}(M'_1|_V + M'_2|_V) \leq 10R_4 \cdot \log^2(d)] \leq |\mathbb{F}|^{-\log^2(n)}.$$

This completes the proof of lemma 27. \square

We now continue with the proof of Lemma 26. Notice that whenever $\text{rank}(M_1|_V + M_2|_V) > 10R_4 \cdot \log^2(d)$ we get by Corollary 7 that $M_1|_V + M_2|_V$ is the unique $\Sigma\Pi\Sigma(2)$ circuit for $f|_V$ of degree $\leq d$. In this case we of course get that $\text{rank}(f) > 10R_4 \cdot \log^2(d)$. As $\Pr [\text{rank}(M'_1|_V + M'_2|_V) \leq 10R_4 \cdot \log^2(d)] \leq |\mathbb{F}|^{-\log^2(n)}$ we get that $\Pr [\text{rank}(f) \leq 10R_4 \cdot \log^2(d)] \leq |\mathbb{F}|^{-\log^2(n)}$. This completes the proof of Lemma 26. \square

Now that we established that (w.h.p.) $\text{rank}(f|_V)$ is high we continue with the proof of Lemma 25. Assuming that $\text{rank}(f|_V) \geq 10R_4 \cdot \log^2(d)$ we get that, w.l.o.g., there are at least $5R_4 \cdot \log^2(d)$ linearly independent linear functions dividing $M_1|_V$ and not $M_2|_V$. In particular there exist $t = 100 \log d$ linearly independent linear functions $\{\ell_i\}_{i \in [t]}$ such that $\prod_{i=1}^t \ell_i$ divides M_1 but no ℓ_i divides M_2 .

We bound the running time in the obvious manner. During the step, we simply have to run over all sets of $t = 100 \log d$ linear functions and for each such set $\{\ell_i\}$ check whether the functions are linearly independent and whether the restriction of f to $V_i = V \cap \{x : \ell_i(x) = 0\}$ completely factorizes to a product of linear functions. Clearly the running time is equal (up to a polynomial factor in $n, |\mathbb{F}|$) to the number of such sets. As the dimension of V is $s = 20R_4 \log^2(d) + \log^2(n)$, the number of sets that we have to consider is $|\mathbb{F}|^{100 \log(d) \cdot s}$, which is quasi-polynomial in n, d and $|\mathbb{F}|$. This completes the proof of Lemma 25. \square

4.2.2 Step 2: Interpolating on a low dimensional subspace II

As we showed in Lemma 25, we can assume that one of the sets of linearly independent linear functions $\{\ell_i\}_{i \in [t]}$ found in Step 1 satisfy that w.l.o.g., $\prod_{i=1}^t \ell_i$ divides $M_1|_V$ but no ℓ_i divides $M_2|_V$. When studying Step 2 we will focus on this set to show that one of the output circuits is indeed $C|_V$.

¹⁶Although the rank is at least $10R_4 \cdot \log^2(d)$ we only consider a subset of the linear functions of dimension exactly $10R_4 \cdot \log^2(d)$.

Before giving the analysis of Step 2 we first explain how to compute $M_2|_V$ from the different $M_2|_{V_i}$ -s (given that the ℓ_i -s satisfy the property above). The idea behind the reconstruction algorithm is the following. We have to reconstruct the set of linear functions appearing in $M_2|_V$. Since a linear function can divide M_2 several times we can speak about the *multiset* of linear functions dividing $M_2|_V$, which we denote with \mathcal{L} . As we have oracle access to each V_i , using the factoring algorithm of Theorem 8, we can assume that we have as input the t multisets $\{\mathcal{L}_i\}_{i=1}^t$, where \mathcal{L}_i is the multiset composed of the linear functions dividing $M_2|_{V_i}$. What the algorithm actually does is to merge the different \mathcal{L}_i -s to get \mathcal{L} . To simplify notations we shall make several assumptions on V and $\{\ell_i\}_{i \in [t]}$ that will be w.l.o.g. and will ease the reading of the algorithm¹⁷. First we assume that

$$V = \{(a_1, \dots, a_s, 0, \dots, 0) : \forall i a_i \in \mathbb{F}\}.$$

Hence, every linear function defined on V is of the form

$$L(v) = \sum_{i=1}^s \alpha_i a_i + \alpha_0,$$

for the vector $v = (a_1, \dots, a_s, 0, \dots, 0)$. We shall also assume that

$$\ell_i(v) = \ell_i((a_1, \dots, a_s, 0, \dots, 0)) = a_i.$$

In other words we assume that ℓ_i is a projection on the i -th coordinate. Notice that as we can apply linear transformations to \mathbb{F}^n , this can be assumed without loss of generality¹⁸. Thus, by our assumption we get that $V_i = \{(a_1, \dots, a_{i-1}, 0, a_{i+1}, \dots, a_s, \dots, 0) : \forall j a_j \in \mathbb{F}\}$. Thus, the multiset \mathcal{L}_i is the multiset obtained after deleting the i -th variable from all the linear functions in the multiset \mathcal{L} . Hence, from our assumptions (that can be made w.l.o.g.) we see that our goal is to reconstruct a multiset of linear functions \mathcal{L} , from its t different projections $\{\mathcal{L}_i\}_{i \in [t]}$ on $s - 1 \geq t - 1$ variables. Algorithm 5 solves exactly this task. In fact the algorithm assumes that we have only t variables (and not s) and is still able to solve the problem. Hence, using Algorithm 5 we are able to reconstruct the set \mathcal{L} and from it get $M_2|_V$.

To explain how Algorithm 5 works we shall use the following terminology. We say that a linear function T has multiplicity k in a multiset \mathcal{T} if there are exactly k linear functions in \mathcal{T} that are linearly dependent on T (that is, that are equal to some multiple of T). The idea behind the algorithm is the following. First it finds a linear function $L \in \mathcal{L}_1$ that has the following property: Assume that L appears k times in \mathcal{L}_1 . Then there exists $i \in \{2, \dots, t\}$ such that $L|_{x_i=0}$ also has multiplicity k in $\mathcal{L}_i|_{x_i=0}$. In other words, L has the property that there is no other linear function $L' \in \mathcal{L}_1$ such that L and L' span together x_i . Note that it is not clear why such a linear function L should exist, however using known lower bounds on

¹⁷If we were to describe the most general case then we had to use the dual basis as in Step 3.

¹⁸Actually, ℓ_i can be an affine function and so should be of the form $a_i + \nu_i$, for some constant ν_i . However, this will not change the algorithm and will only add unnecessary complications to the presentation.

Locally Decodable codes (see Appendix B) we can prove its existence. Now, given such L , we note that by considering \mathcal{L}_i there are exactly k linear functions $\{L_i\}_{i=1}^k$ that, after erasing their first coordinate, are equal to (a constant multiple of) $L|_{x_i=0}$. Thus, we are assured that there are exactly k linear function $\{\ell_i\}_{i=1}^k$ in \mathcal{L} , such that $\ell_i|_{x_1=0} = L$ and $\ell_i|_{x_i=0} = L_i$. In particular, to get ℓ_i we simply add the first coordinate of L_i to L , for each $i \in [k]$ (after making the appropriate normalization). Now, we can remove the k projections of $\{\ell_i\}_{i=1}^k$ from the different \mathcal{L}_i -s and repeat the algorithm until no linear function is left. From this description it is clear that the main technical point is proving that such a linear function $L \in \mathcal{L}_1$ exists. We now give the formal description of Algorithm 5 and then we analyze it. After that it will be clear how to analyze Step 2.

Algorithm 5 Gluing the different $M_2|_{V_i}$ -s together

Input: Multisets of linear functions $\mathcal{L}_i = \mathcal{L}|_{x_i=0}$, for $i = 1 \dots t$, containing $m < \exp((t - 1)/60 - 1)$ linear functions each.

Output: a multiset $\tilde{\mathcal{L}}$.

Set $\tilde{\mathcal{L}} \leftarrow \emptyset$.

1. Find a linear function $L \in \mathcal{L}_1$ and an index $1 < i \leq t$ such that the following holds: denote with k the multiplicity of L in \mathcal{L}_1 . Then the number of $L' \in \mathcal{L}_i$ such that $L'|_{x_1=0} \sim L|_{x_i=0}$ is exactly k (and in particular, $L|_{x_i=0}$ is not a constant).
 2. Denote with $\{L_j\}_{j=1}^k \subset \mathcal{L}_i$ those k linear functions. Let $\{c_j\}_{j=1}^k$ be such that $L|_{x_i=0} = c_j \cdot L_j|_{x_1=0}$. Let $a_{j,1}$ be the coefficient of x_1 in the linear function $c_j \cdot L_j$. Let $\ell_j \triangleq L + a_{j,1} \cdot x_1$, for $j = 1 \dots k$.
 3. Set $\tilde{\mathcal{L}} \leftarrow \tilde{\mathcal{L}} \cup \{\ell_j\}_{j \in [k]}$.
 4. For each $l \in [t]$, set $\mathcal{L}_l \leftarrow \mathcal{L}_l \setminus \{\ell_j|_{x_l=0}\}_{j=1}^k$ (more accurately, remove from \mathcal{L}_l k linear functions ℓ'_1, \dots, ℓ'_k such that $\ell'_j = \ell_j|_{x_l=0}$).
 5. If the \mathcal{L}_i -s are empty then return $\tilde{\mathcal{L}}$, otherwise go to Step 1.
-

The following theorem proves correctness and gives the analysis of Algorithm 5.

Theorem 29. *Let $\mathcal{L} = \{L_i\}_{i=1}^m$ be a multiset of linear functions in $t > 60 \log m + 61$ variables. I.e., a linear function may appear more than once in \mathcal{L} . Denote with $\mathcal{L}_i \triangleq \mathcal{L}|_{x_i=0}$ the multiset resulting from \mathcal{L} after removing the i -th variable from all the linear functions in it. Then, given the multisets $\{\mathcal{L}_i\}_{i \in [t]}$, Algorithm 5 reconstructs a multiset $\tilde{\mathcal{L}}$ such that $\prod_{\ell \in \tilde{\mathcal{L}}} \ell = c \cdot \prod_{\ell' \in \mathcal{L}} \ell'$, for some non-zero constant c , in time polynomial in m .*

Proof. First we show that if a linear function L was found in Step 1, and its multiplicity in \mathcal{L}_1 is k , then there are exactly k linear functions $\{\tilde{\ell}_j\}_{j \in [k]}$ in \mathcal{L} that satisfy $\tilde{\ell}_j|_{x_1=0} \sim L$, for each $j \in [k]$. Moreover, these linear functions are equal, up to a multiplicative factor, to the linear functions $\{\ell_j\}_{j \in [k]}$ that were found in Step 2 of the algorithm. Namely, possibly after

reordering it holds that $\tilde{\ell}_j \sim \ell_j$, for $j \in [k]$. Indeed, let L and i be the linear function and index that were found in Step 1 (of Algorithm 5). Denote with k the multiplicity of L in \mathcal{L}_1 and let $\{\tilde{\ell}_j\}_{j \in [k]} \subset \mathcal{L}$, be such that $\tilde{\ell}_j|_{x_1=0} \sim L$. From the definition of i these are exactly the functions in \mathcal{L} that satisfy that $\tilde{\ell}_j|_{x_1=0, x_i=0} \sim L|_{x_i=0}$. We therefore have the following. Let a_j be such that $a_j \cdot \tilde{\ell}_j|_{x_1=0} = L$. Let c_j and L_j (for $j \in [k]$) be as in Step 2. It must be the case that (possibly after reordering) $a_j \cdot \tilde{\ell}_j|_{x_i=0} = c_j \cdot L_j$. In particular if we define $\ell_j = \alpha_{j,1} \cdot x_1 + L$, where $\alpha_{j,1}$ is the coefficient of x_1 in $c_j \cdot L_j$ then it must be the case that $\ell_j \sim \tilde{\ell}_j$ as required (note that a different way of defining ℓ_j is by letting $\ell_j = c_j \cdot L_j + \alpha_i \cdot x_i$, where α_i is the coefficient of x_i in L). In particular this shows that Step 3 adds to $\tilde{\mathcal{L}}$ linear functions that are equal to the $\tilde{\ell}_j$ -s up to a constant factor and Step 4 removes the “images” of those functions from the different \mathcal{L}_i -s. Thus, if we can always find such L and i then we are guaranteed that the output will be a set $\tilde{\mathcal{L}}$ for which there is a 1-1 correspondence between the linear functions in \mathcal{L} and those in $\tilde{\mathcal{L}}$ that matches functions that are equal up to a (non-zero) constant factor. This completes the first part of our proof.

We now prove that if $m < \exp((t-1)/60 - 1)$ then such L and i exist. Assume for a contradiction that no such L and i exist. Then it must be the case that for every $L \in \mathcal{L}_1$ and i there is $L' \in \mathcal{L}_1$ such that $x_i \in \text{span}(L, L')$. In other words, if we consider the mapping $E : \mathbb{F}^{t-1} \rightarrow \mathbb{F}^m$ that maps (x_2, \dots, x_t) to $\{L(x_2, \dots, x_t)\}_{L \in \mathcal{L}_1}$, then we have that for every $j \in \{2, \dots, t\}$ there are, say, at least $m/3$ disjoint pairs of linear functions $L, L' \in \mathcal{L}_1$ such that $x_j \in \text{span}(L, L')$. In other words, we can reconstruct x_j by looking at one of $m/3$ disjoint pairs of indices in the mapping E . A mapping that satisfies this property is called a 2-query locally decodable codes and it is known (Theorem 33) that in such a code it must be the case that $m \geq \exp((t-1)/60 - 1)$. However, we assumed that $m < \exp((t-1)/60 - 1)$ and so we reached a contradiction. Hence a linear function L and an index i can be found. For completeness we discuss locally decodable codes in Appendix B.

The claim regarding the running time of Algorithm 5 is clear. This completes the proof of Theorem 29. \square

To conclude, from Theorem 29 and the discussion at the beginning of Section 4.2.2 we see that in Step 2 of Algorithm 4, for one of the sets $\{\ell_i\}_{i \in [t]}$ we manage to reconstruct the multiplication gate $M_2|_V$. To be more accurate, note that for the multiset $\tilde{\mathcal{L}}$ computed in Algorithm 5 we may have that $\prod_{\ell \in \tilde{\mathcal{L}}} \ell = c \cdot \prod_{\ell' \in \mathcal{L}} \ell'$ for some non-zero constant c . However, c can be easily found by comparing $\prod_{\ell \in \tilde{\mathcal{L}}} \ell|_{x_1=0}$ to $M_2|_{x_1=0}$. Therefor, we can multiply some $\ell \in \tilde{\mathcal{L}}$ by c to get that $\prod_{\ell \in \tilde{\mathcal{L}}} \ell = \prod_{\ell' \in \mathcal{L}} \ell' = M_2|_V$.

Now, given $M_2|_V$ we can use the oracle to f to factor $f|_V - M_2|_V$ and get $M_1|_V$. In particular we compute the circuit $C|_V = M_1|_V + M_2|_V$ in Step 2. We now notice that if V is such that $\text{rank}(f|_V) \geq R_4 \log^2(d)$ (which happens w.h.p. according to Lemma 26) then by Corollary 6 this is the only possible representation for $f|_V$ (as a $\Sigma\Pi\Sigma(2)$ circuit) and so the circuit computed at the end of this step is indeed $M_1|_V + M_2|_V$. The running time of Step 2 is equal to the number of different sets $\{\ell_i\}_{i \in [t]}$ considered times the running times of Algorithm 5 and the factoring algorithm of Theorem 8. Thus, we just proved the following

theorem.

Theorem 30. *Step 2 of Algorithm 4 runs in time $\exp(\text{poly}(\log n, \log d, \log |\mathbb{F}|))$ and if V is such that $\text{rank}(f|_V) \geq R_4 \log^2(d)$ then the circuit computed at the end of the step is $C|_V = M_1|_V + M_2|_V$.*

4.2.3 Step 3: Lifting

We first give the intuition behind the way that the algorithm combines the different circuits for $f|_{W_i}$. For convenience let us assume w.l.o.g., again, that

$$V = \{v = (a_1, \dots, a_s, 0, \dots, 0) \mid \forall i a_i \in \mathbb{F}\}.$$

Also assume w.l.o.g. that for $1 \leq i \leq n - s$ we have that

$$W_i = \{w = (a_1, \dots, a_s, 0, \dots, 0, a_{s+i}, 0, \dots, 0) \mid \forall j a_j \in \mathbb{F}\}.$$

Note that as we are working with the v_i -s and their dual basis $\{z_i\}$ then we can make this assumption. Let us assume that $\text{rank}(f|_V) \geq R_4 \cdot \log^2(d)$. Theorem 30 now guarantees that the algorithm computes correctly the circuits $C_i = M_1|_{W_i} + M_2|_{W_i}$, for $s < i \leq n$. Assume that our original $\Sigma\Pi\Sigma(2)$ for f has the following form

$$f(\bar{x}) = \prod_{i=1}^d L_i^{(1)}(\bar{x}) + \prod_{i=1}^d L_i^{(2)}(\bar{x}),$$

where $M_1 = \prod_{i=1}^d L_i^{(1)}(\bar{x})$ and $M_2 = \prod_{i=1}^d L_i^{(2)}(\bar{x})$. Denote

$$L_i^{(k)} = \sum_{j=1}^n \alpha_{i,j}^{(k)} x_j + \alpha_{i,0}^{(k)},$$

for $k = 1, 2$. If $\text{rank}(f|_V) > R_4 \cdot \log^2(d)$ then by Theorem 30 we get that

$$\begin{aligned} f|_{W_i} &= \prod_{k=1}^d L_k^{(1)}(\bar{x})|_{W_i} + \prod_{k=1}^d L_k^{(2)}(\bar{x})|_{W_i} \\ &= \prod_{k=1}^d \left(\sum_{j=1}^s \alpha_{k,j}^{(1)} x_j + \alpha_{k,s+i}^{(1)} x_{s+i} + \alpha_{k,0}^{(1)} \right) + \prod_{k=1}^d \left(\sum_{j=1}^s \alpha_{k,j}^{(2)} x_j + \alpha_{k,s+i}^{(2)} x_{s+i} + \alpha_{k,0}^{(2)} \right). \end{aligned}$$

Therefore all that we have to do is to find all the linear functions of the form $\sum_{j=1}^s \alpha_{k,j}^{(1)} x_j + \alpha_{k,s+i}^{(1)} x_{s+i} + \alpha_{k,0}^{(1)}$, for $i \in [n - s]$, and glue them together to get all the linear functions $\{\sum_{j=1}^n \alpha_{k,j}^{(1)} x_j\}_k$ (similarly with $\sum_{j=1}^s \alpha_{k,j}^{(2)} x_j + \alpha_{k,s+i}^{(2)} x_{s+i} + \alpha_{k,0}^{(2)}$ and $\{\sum_{j=1}^n \alpha_{k,j}^{(2)} x_j\}_k$). To do so we must be sure that there are no two linearly independent linear functions in f that their restrictions to V are linearly dependent. Note that if such a pair exists, e.g. if $\ell|_V \sim \ell'|_V$,

where ℓ and ℓ' are linearly independent, then for some W_i and constants $\alpha, \alpha', \alpha''$ such that $\alpha \neq \alpha''$, the circuit C_i will contain two different functions $\tilde{\ell} = \ell + \alpha \cdot x_i$ and $\tilde{\ell}' = \ell' + \alpha' \cdot x_i \sim \ell + \alpha'' \cdot x_i$ and we will not know which of them to attach to ℓ and which to ℓ' when combining appropriate linear functions from the different C_i -s.

Now, if no such pair exists then it is very easy to combine the different circuits C_i into one circuit C' for f : for every linear function ℓ in C , that does not divide both its multiplication gates, there is a unique linear function ℓ_i in C_i such that their restrictions to $x_i = 0$ are linearly dependent (note that by Lemma 28 we can assume that $\text{g.c.d.}(M_1|_V, M_2|_V) = \text{g.c.d.}(M_1, M_2)|_V$). Hence, if ℓ divides both multiplication gates in C then there is a unique ℓ_i that divides both multiplication gates in C_i , and the continuation is the same for this case as well). Therefore there is only one way of generating a linear function L such that $\ell|_V \sim \ell$ and $\ell'|_{x_i=0} \sim \ell_i$, for every i . This explanation shows that in Step 3 we combine the different circuits in the only possible way. Moreover it is clear that combining the C_i -s to a single circuit can be done efficiently. It only remains to justify the assumption that there are no two linearly independent linear functions in f that their restrictions to V are linearly dependent.

To see that we notice that as we picked V to be a random subspace of dimension $\Omega(\log^2(d))$ then Lemma 21 assures us that with probability greater than $1 - \exp(-\log^2(n))$ no two linearly independent linear functions from C become linearly dependent when restricted to V . Thus, w.h.p. we can complete this step. Notice that by the above explanation it is also very easy to verify that indeed no two linearly independent linear functions from the circuit for f became dependent on V . The following theorem summarizes what we have shown in this section.

Theorem 31. *Step 3 of Algorithm 4 runs in time $\text{poly}(n, d, \log |\mathbb{F}|)$ and w.h.p. over the choice of V outputs the unique $\Sigma\Pi\Sigma(2)$ circuit for f .*

Thus, Algorithm 4 outputs with high probability over the choice of V the unique $\Sigma\Pi\Sigma(2)$ for f . We now show how to combine Algorithms 3 and 4 to get Theorem 2.

4.3 Completing the proof of Theorem 2

We note that if we know $\text{rank}(f)$ then the algorithms described in Sections 4.1 and 4.2 give the required result. As we don't know what the rank is we run the high rank algorithm for every rank between $10R_4 \cdot \log^2(d)$ and d . Once the algorithm output a $\Sigma\Pi\Sigma(2)$ circuit we verify that its rank is indeed larger than $10R_4 \cdot \log^2(d)$. If this is the case then we stop and output that circuit. Theorem 31 guarantees that w.h.p. we have found the unique $\Sigma\Pi\Sigma(2)$ circuit C' that computes f . If none of the executions of the high rank algorithm resulted in a $\Sigma\Pi\Sigma(2)$ circuit then we run the low rank algorithm for each rank between 1 and $10R_4 \cdot \log^2(d)$.

To analyze the error probability of the algorithm we note that the possible errors are in the choice of V and in the factoring algorithm. Indeed, when we are at the "correct" rank

and V is such that the rank of $f|_V$ is the same as $\text{rank}(f)$ and no two linearly independent linear functions from C become linearly dependent when restricted to V then we are assured that we compute a correct representation for f . Recall that V is “bad” for us with probability $1/p(n, d)$ where p is some quasi-polynomial function of n and d . In addition we can repeat the factoring algorithm of Theorem 8 polynomially many times to reduce the error probability. Thus the over all error is quasi-polynomially small (and can be further reduced to be exponentially small by repeating the algorithm several times). This completes the proof of the theorem.

Acknowledgements

I would like to thank Amir Yehudayoff and Nader Bshouty for many helpful discussions, Erich Kaltofen and Joachim von zur Gathen for answering my questions regarding factorization algorithms and Avi Wigderson and the anonymous reviewers for helpful comments. Thanks to Oded Goldreich and Ran Raz for their encouragement. The seed for this work was planted in the BIRS workshop “Recent Advances in Computation Complexity”. I thank the organizers for inviting me and BIRS for hosting the workshop.

References

- [BB98] D. Bshouty and N. H. Bshouty. On interpolating arithmetic read-once formulas with exponentiation. *J. of Computer and System Sciences*, 56(1):112–124, 1998.
- [BBB⁺00] A. Beimel, F. Bergadano, N. H. Bshouty, E. Kushilevitz, and S. Varricchio. Learning functions represented as multiplicity automata. *JACM*, 47(3):506–530, 2000.
- [BBTV97] F. Bergadano, N. H. Bshouty, C. Tamon, and S. Varricchio. On learning branching programs and small depth circuits. In *Proceedings of the 3rd European Conference on Computational Learning Theory*, volume 1208 of *LNAI*, pages 150–161, 1997.
- [BBV96] F. Bergadano, N. H. Bshouty, and S. Varricchio. Learning multivariate polynomials from substitution and equivalence queries. *ECCC*, 3(8), 1996.
- [BC98] N. H. Bshouty and R. Cleve. Interpolating arithmetic read-once formulas in parallel. *SIAM J. on Computing*, 27(2):401–413, 1998.
- [BHH95] N. H. Bshouty, T. R. Hancock, and L. Hellerstein. Learning boolean read-once formulas with arbitrary symmetric and constant fan-in gates. *Journal of Computer and System Science*, 50:521–542, 1995.

- [BOT88] M. Ben-Or and P. Tiwari. A deterministic algorithm for sparse multivariate polynomial interpolation. In *Proceedings of the 20th Annual STOC*, pages 301–309, 1988.
- [DS06] 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, 2006.
- [FK06] L. Fortnow and A. R. Klivans. Efficient learning algorithms yield circuit lower bounds. In *Proceeding of the 19th annual COLT*, pages 350–363, 2006.
- [GKS94] D. Grigoriev, M. Karpinski, and M. F. Singer. Computational complexity of sparse rational interpolation. *SIAM J. on Computing*, 23(1):1–11, 1994.
- [GKST06] O. Goldreich, H. J. Karloff, L. J. Schulman, and L. Trevisan. Lower bounds for linear locally decodable codes and private information retrieval. *Computational Complexity*, 15(3):263–296, 2006.
- [HH91] T. R. Hancock and L. Hellerstein. Learning read-once formulas over fields and extended bases. In *Proceedings of the 4th Annual COLT*, pages 326–336, 1991.
- [Kal85] E. Kaltofen. Polynomial-time reductions from multivariate to bi- and univariate integral polynomial factorization. *SIAM J. on computing*, 14(2):469–489, 1985.
- [Kal95] E. Kaltofen. Effective Noether irreducibility forms and applications. *J. of Computer and System Sciences*, 50(2):274–295, 1995.
- [Kha95] M. Kharitonov. Cryptographic lower bounds for learnability of boolean functions on the uniform distribution. *J. of Computer and System Sciences*, 50(3):600–610, 1995.
- [KL01] M. Krause and S. Lucks. Pseudorandom functions in TC0 and cryptographic limitations to proving lower bounds. *Computational Complexity*, 10(4):297–313, 2001.
- [KS96] M. Karpinski and I. Shparlinski. On some approximation problems concerning sparse polynomials over finite fields. *Theoretical Computer Science*, 157(2):259–266, 1996.
- [KS01] A. Klivans and D. Spielman. Randomness efficient identity testing of multivariate polynomials. In *Proceedings of the 33rd Annual 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.
- [KS08] Z. Karnin and A. Shpilka. Reconstruction of generalized depth-3 arithmetic circuits with bounded top fan-in. Manuscript, 2008.

- [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.
- [KT00] J. Katz and L. Trevisan. On the efficiency of local decoding procedures for error-correcting codes. In *Proceedings of the 32nd ACM symposium on Theory of computing*, pages 80–86. ACM press, 2000.
- [KV94] M. J. Kearns and L. G. Valiant. Cryptographic limitations on learning boolean formulae and finite automata. *J. ACM*, 41(1):67–95, 1994.
- [Man95] Y. Mansour. Randomized interpolation and approximation of sparse polynomials. *SIAM J. on computing*, 24(2):357–368, 1995.
- [NR04] M. Naor and O. Reingold. Number-theoretic constructions of efficient pseudo-random functions. *J. ACM*, 51(2):231–262, 2004.
- [OGM86] S. Goldwasser O. Goldreich and S. Micali. How to construct random functions. *J. ACM*, 33(4):792–807, 1986.
- [RR97] A. A. Razboeov and S. Rudich. Natural proofs. *J. of Computer and System Sciences*, 55(1):24–35, 1997.
- [Shp07] A. Shpilka. Interpolation of depth-3 arithmetic circuits with two multiplication gates. In *Proceedings of the 39th Annual STOC*, pages 284–293, 2007.
- [SS96] R. E. Schapire and L. M. Sellie. Learning sparse multivariate polynomials over a field with queries and counterexamples. *J. of Computer and System Sciences*, 52(2):201–213, 1996.
- [SV08] A. Shpilka and I. Volkovich. Read-once polynomial identity testing. In *Proceedings of the 40th Annual STOC*, pages 507–516, 2008.
- [SW01] A. Shpilka and A. Wigderson. Depth-3 arithmetic circuits over fields of characteristic zero. *Computational Complexity*, 10(1):1–27, 2001.

A Brute force interpolation

In this section we show how to interpolate a polynomial that can be written as a polynomial in a few linear functions. In fact we need to consider a slightly more complicated scenario as described in the following lemma.

Lemma 32. Let¹⁹ $h(x_1, \dots, x_s) = \text{Lin}(h) \cdot Q(\ell_1, \dots, \ell_k)$ be a polynomial, where Q is a polynomial and $\{\ell_i\}_{i \in [k]}$ are linear functions. Let $d = \deg(h) < |\mathbb{F}|$. Then, there is a deterministic algorithm that given oracle access to h and $\text{Lin}(h)$, d and the set of linear functions $\{\ell_i\}_{i \in [k]}$, finds Q . The running time of the algorithm is $\text{poly}(|\mathbb{F}|^s, d^k)$.

Proof. Consider a generic degree d polynomial Q' in the ℓ_i -s. Such a polynomial has $< (d+1)^k$ monomials of the form $\prod_{i=1}^k \ell_i^{e_i}$, such that $\sum_{i=1}^k e_i \leq d$. Thus we have a number of unknown coefficients which we can find by a simple interpolation procedure that we now describe: Let $\ell_{k+1}, \dots, \ell_s$ be linear functions such that²⁰ ℓ_1, \dots, ℓ_s form a basis to the space of linear functions over \mathbb{F}^s . Let

$$Q'(y_1, \dots, y_k) = \sum_{\{\bar{e}=(e_1, \dots, e_k): \sum e_i \leq d\}} c_{\bar{e}} \cdot \prod y_i^{e_i}. \quad (11)$$

Next we represent $\text{Lin}(h)$ as a polynomial in the ℓ_i -s. Let $P_{\text{Lin}}(y_1, \dots, y_s)$ be a polynomial satisfying

$$\text{Lin}(h) = P_{\text{Lin}}(\ell_1, \dots, \ell_s) = \sum_M \alpha_M \cdot M(\ell_1, \dots, \ell_s), \quad (12)$$

where the sum is over all monomials M of degree at most d . Note that as we have oracle access to $\text{Lin}(h)$, it is easy to find the α_M -s (by the usual interpolation process). We get that h can be written as

$$h = P_{\text{Lin}}(\ell_1, \dots, \ell_s) \cdot Q'(\ell_1, \dots, \ell_k) = \sum_M \gamma_M \cdot M(\ell_1, \dots, \ell_s), \quad (13)$$

where each coefficient γ_M is a linear function in the $c_{\bar{e}}$ -s. By querying h on all the points in \mathbb{F}^s and using the standard interpolation method we can easily find all the coefficients γ_M , and from them we can get the coefficients $c_{\bar{e}}$ -s by solving a system of linear equations. We note that once we know the γ_M -s there is a unique solution to the $c_{\bar{e}}$ -s (in case that such a solution exists). The reason is that if two different solutions exist then they give rise to two polynomials $P_{\text{Lin}} \cdot Q_1$ and $P_{\text{Lin}} \cdot Q_2$, that are equal over \mathbb{F}^s . However, the degree of each of the polynomials is at most d , which is smaller than the size of the field that we are working with, and so the two polynomials must be the same. In particular we have that $Q' = Q$. \square

B Locally decodable codes

In this section we define the notion of locally decodable codes and state the result that we need. The reader interested in a more complete background is referred to [KT00, GKST06]. We start with the definition of locally decodable codes. We fix in advance some of the

¹⁹In Section 4.1.2 we needed to interpolate $\text{sim}(f)|_V$ for an s -dimensional space V , so we assume that h is a polynomial in s variables.

²⁰We assume w.l.o.g. that ℓ_1, \dots, ℓ_k are linearly independent.

parameters to constants in order to simplify the definition. Let $E : \mathbb{F}^t \rightarrow \mathbb{F}^n$ be a linear map. We say that E is a 2-query locally decodable code if there exists a probabilistic oracle machine A such that:

- A makes at most 2 queries (non-adaptively) to the oracle.
- For every $x \in \mathbb{F}^t$, for every $y \in \mathbb{F}^n$ with $\Delta(y, E(x)) < n/10$, and for every $i \in [t]$, we have $\Pr[A^y(i) = x_i] \geq 2/3$, where the probability is taken over the internal coin tosses of A , and $\Delta(\cdot, \cdot)$ is the Hamming distance function.

The following theorem of [DS06] gives a lower bound on the length of such locally decodable codes over arbitrary fields.

Theorem 33 (Theorem 1.2 of [DS06]). *Let \mathbb{F} be a field, and let $E : \mathbb{F}^t \rightarrow \mathbb{F}^n$ be a linear 2-query locally decodable code, as in the definition above, then $n \geq 2^{\frac{t}{60}-1}$.*

We note that Goldreich et al [GKST06] were the first to prove a lower bound for linear locally decodable codes with 2 queries, however for large fields their result is not optimal. In particular, [GKST06] proved a lower bound of the form $n \geq 2^{\Omega(t) - \log |\mathbb{F}|}$ on the length of such codes over \mathbb{F} , which deteriorates with the field size.