The Partial Derivative method in Arithmetic Circuit Complexity

A thesis submitted in partial fulfillment of the requirements for the award of the degree of

Master of Science

by

Yadu Vasudev

Junior Research Fellow
Theoretical Computer Science
The Institute of Mathematical Sciences
Chennai - 600113

Homi Bhabha National Institute

April 2010

Declaration

I declare that the thesis titled *The Partial Derivative method in Arithmetic Circuit Complexity* is a record of the work done by me during the period January 2010 to April 2010 under the supervision of Prof. V. Arvind. This work has not been submitted earlier as a whole or in part for a degree, diploma, associateship or fellowship at this institute or any other institute or university.

Yadu Vasudev

Certificate

Certified that the work contained in the thesis entitled THE PARTIAL DERIVATIVE METHOD IN ARITHMETIC CIRCUIT COMPLEXITY, by **Yadu Vasudev**, has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

V.Arvind
Theoretical Computer Science Group
The Institute of Mathematical Sciences, Chennai

ACKNOWLEDGEMENTS

I would like to thank my advisor V.Arvind for his guidance during the course of this thesis. He has always been there to patiently answer all my questions.

Let me also thank all my friends here with whom I had a wonderful time during the last two years. My parents have always been supportive of all my decisions and I thank them for being there for me.

Abstract

In this thesis we survey the technique of analyzing the partial derivatives of a polynomial to prove lower bounds for restricted classes of arithmetic circuits. The technique is also useful in designing algorithms for learning arithmetic circuits and we study the application of the method of partial derivatives in this setting. We also look at polynomial identity testing and survey an efficient algorithm for identity testing certain classes of arithmetic circuits.

The dimension of the vector space of partial derivatives of a polynomial will be the quantity of interest in this thesis. Lower bounds for an explicit polynomial against a class of circuits is obtained by comparing the dimension of the space of partial derivatives of the polynomial and the dimension of the space of polynomials computed by that class of circuits. The efficiency of the learning algorithms that we survey also depends on the dimension of partial derivatives of the class of functions. We also note the connections between the problems of proving lower bounds, designing efficient learning algorithms and identity testing arithmetic circuits.

Contents

Chapter 1

Introduction

1.1 Lower bounds in Complexity Theory

Obtaining limits on the computational power of models and proving lower bounds on the resources used by them is an important problem in Computational Complexity. The holy grail in the area is the relationship between the computational power of P and NP. Proving time and space lower bounds for Turing machines seem to be very hard, and the non-trivial separations here are given by the time and space hierarchy theorem.

Another approach to resolving the P versus NP problem has been via proving lower bounds for boolean circuits. These are non-uniform models of computation where you have a separate circuit for each input length. Languages accepted by polynomial time turing machines have a polynomial sized boolean circuit computing them. Hence showing that a language in NP requires super-polynomial sized boolean circuit would give a separation between the two classes.

The works of Furst *et.al* and Håstad[FSS84, Hås86] showed that any constant depth circuit computing the parity of the input bits require exponential size. This result separated the class of AC^0 circuits from NC^1 circuits. Razborov and Smolensky[Raz87, Smo87] used algebraic techniques to show that “small” boolean circuits with MOD_q gates for a prime q cannot compute the sum of the inputs modulo a prime $p \neq q$. The work of Razborov and Rudich[RR97] showed that the methods employed in the above proofs were unlikely to give a separation of P and NP.

The model of computation that we will be interested in this thesis are *arithmetic circuits*. These are more constrained models of computation since boolean circuits are arithmetic circuits over $\mathbb{GF}[2]$ with the additional identity $x^2 = x$[Val92]. Arithmetic circuits are natural and succinct model for representing polynomials.

1.2 Lower Bounds in Arithmetic Circuit Complexity

Even in the case of such restricted classes of circuits, non-trivial lower bounds are hard to come by. The best known lower bound in the case of general arithmetic circuits is the lower bounds of Baur and Strassen[Str73, BS83] which we survey in the next chapter. The difficulty in proving strong lower bounds for general circuits has led to analysing other constrained classes of arithmetic circuits, where there has been more success.

In the case of depth 3 circuits, Nisan and Wigderson[NW97] proved exponential lower bounds for the symmetric polynomial and iterated matrix multiplication when the circuit is homogeneous. In the case of general depth 3 circuits, Shpilka and Wigderson[SW01] give lower bounds for symmetric polynomial, trace of iterated matrix multiplication and determinant. Grigoriev and Razborov[GR98] have given exponential lower bounds for certain explicit symmetric functions and certain generalizations of MOD_q and MAJ over any finite field. An exponential lower bound for the determinant function over arbitrary finite fields have been given by Grigoriev and Karpinsky[GK98].

Nisan[Nis91] initiated the study of non-commutative models of computation in the hope of getting more insight about commutative circuits. He was able to prove exponential lower bounds for the permanent and determinant polynomials in the case of non-commutative formulas. These ideas have been extended by Chien and Sinclair[CS07] to prove lower bounds for the permanent over matrix algebras.

In the case of multilinear circuits, Raz[Raz09] has proved exponential lower bounds for the determinant and permanent for multilinear formulas. Raz[Raz04] give a separation between bounded fan-in $\log$-depth multilinear circuits and bounded fan-in $\log^2$-depth multilinear circuits. Nisan and Wigderson[NW97] have proved lower bounds for the depth in the case of multilinear circuits.

Here we have cited only those lower bounds which are in some measure related to the results that we are surveying in this thesis.

1.3 The Goal of this thesis

The goal of this thesis is to understand the connections between arithmetic circuit lower bounds, polynomial idetity testing and learning theory questions. This is motivated by a heuristic observation that proving circuit lower bounds is easier that polynomial identity testing which in turn is easier than designing efficient learning algorithms.

The thesis is organized as follows: In the next chapter, we set up the basic definitions of arithmetic circuits and the lower bounds technique that we will be interested in. We will then look at applying these techniques to get lower bounds for non-commutative formulas and restricted classes of commutative circuits. We will also survey the classical results of Baur and

Strassen[Str73, BS83].

In the third chapter we analyse learning algorithms of Klivans and Shpilka[KS06] and Beimel *et.al*[BBB$^+$00]. We will look at the connection between the rank of the Hankel matrix of the function and the dimension of the space of partial derivatives.

The fourth chapter is related to the problem of polynomial identity testing. We survey the result of Raz and Shpilka[RS05] which gives a deterministic algorithm for identity testing non-commutative formulas. As a consequence, they also get an identity testing algorithm for depth 3 set-multilinear circuits.

Chapter 2

The Partial Derivative Method

2.1 Notations and Definitions

In this chapter we introduce the definition and notation that will be used in this thesis. The model of computation we consider are arithmetic circuits over a field $\mathbb{F}$. Formally we have

Definition 2.1. *An* arithmetic circuit *on n variables is a vertex-labelled DAG. All the nodes of in-degree 0 are labelled by elements of the set $\{x_1, \cdots, x_n\}$ or by constants from the field $\mathbb{F}$. All other nodes are labelled by $+$ or $\times$. A node is designated as the output node and labelled* Out.

We can define the polynomial computed by the circuit inductively as follows:

- The polynomial computed by a node labelled by a scalar is that constant itself. A node labelled by a variable x_i computes the polynomial x_i.
- Let g be a $+$ gate with inputs $g_1, \cdots, g_m$. The polynomial computed by g is
$$P_g = \sum_{i=1}^{m} P_{g_i}$$

- Let g be a $\times$ gate with inputs $g_1, \cdots, g_m$. The polynomial computed by g is

$$P_g = \prod_{i=1}^{m} P_{g_i}$$

The computations are done over a fixed field $\mathbb{F}$ and we say that the circuit computes a polynomial over the field $\mathbb{F}$. We will also refer to the vertices in the DAG as gates of the circuit. An arithmetic circuit is characterized by the following parameters.

Definition 2.2. *The* size *of a circuit is the number of vertices in the DAG(In some cases we will count the number of edges in the circuit as the size). The* depth *of a circuit is the longest path from* Out *to an input variable.* Fan-in *of a gate is the in-degree of the vertex in the DAG. Correspondingly, the* fan-out *is the out-degree of the vertex.*

A related model that will also be considered is that of an arithmetic branching program.

Definition 2.3 ([Nis91]). *An* arithmetic branching program(ABP) *is a layered DAG with a source* s *and sink* t*. The vertices of the graph are partitioned into "layers" from 0 to* d *and the edges only go from layer* i *to* $i+1$*.* d *is called the degree of the ABP. The source is the only vertex at layer 0 and the sink is the only vertex at layer* d*. The edges of the circuit are labelled by linear forms(i.e* $\sum_i c_i x_i$*). The size of the ABP is the number of vertices in the DAG.*

The polynomial f computed by the ABP is obtained by multiplying the linear forms over each path from s to t and summing it over all $s-t$ paths in the ABP,i.e

$$f(x_1, \ldots, x_n) = \sum_{\text{all } s-t \text{ paths } p} \prod_{i=1}^{d} l_i^p(x_1, \ldots, x_n)$$

where $l_1^p, \dots, l_d^p$ are the linear forms along the path p.

For a polynomial $f \in \mathbb{F}[x_1, \dots, x_n]$, we denote $\partial(f)$ as the set of all partial derivatives of f. We will denote by $\partial_d(f)$, the set of partial derivatives of f of order d. For set of polynomials $P \subseteq \mathbb{F}[x_1, \dots, x_n]$, by $\partial(P)$ we will denote the set of all partial derivatives of P. The definition for partial derivatives in the case of polynomials $f \in \mathbb{F}\langle x_1, \dots, x_n \rangle$ is slightly different and is as follows

Definition 2.4. *For a polynomial* $f \in \mathbb{F}\langle x_1, \dots, x_n \rangle$, *let* $f = \sum_{i=1}^{n} x_i f_i(x_1, \dots, x_n)$, *where* $f_i \in \mathbb{F}\langle x_1, \dots, x_n \rangle$, *then* $\frac{\partial(f)}{\partial(x_i)} = f_i$. *Similarly* $f_i = \sum_{j=1}^{n} x_j f_{i,j}(x_1, \cdots, x_n)$. *Thus* $\frac{\partial^2 f}{\partial x_k \partial x_i} = f_{i,k}$.

Let us look at an example to explain this definition

Example 2.5. *Let* $f(x_1, x_2, x_3) = x_1^2 x_2 + x_2 x_1 + x_1 x_3^2 x_2 + x_3^2 x_2^2 + x_2^2 x_3 x_1 + x_3 x_1$. *The we can write* $f(x_1, x_2, x_3) = x_1(x_2 + x_3^2 x_2) + x_2(x_1 + x_2 x_3 x_1) + x_3(x_3 x_2^2 + x_1)$. *By the earlier definition*

$$\begin{aligned}
\frac{\partial f}{\partial x_1} &= x_2 + x_3^2 x_2 \\
\frac{\partial f}{\partial x_2} &= x_1 + x_2 x_3 x_1 \\
\frac{\partial f}{\partial x_3} &= x_3 x_2^2 + x_1, \ \ \textit{and} \\
\frac{\partial^2 f}{\partial x_2 \partial x_1} &= 1 \\
\frac{\partial^2 f}{\partial x_3 \partial x_1} &= x_3 x_2
\end{aligned}$$

The complexity measure that will be used throughout this thesis is the dimension of the space spanned by the partial derivatives, except for a small section at the end of this chapter. Formally, we have

Definition 2.6. *For a set of polynomials* $P \in \mathbb{F}[x_1, \dots, x_n]$, *we denote by* $\dim(P)$ *the dimension of the vector space spanned by the polynomials in*

P. *In particular* $\dim(\partial(f))$ *denotes the dimension of the space spanned by the partial derivatives of* f. *We denote by* $\dim(\partial_d(f))$*, the dimension of the space spanned by the partial derivatives of* f *of order* d.

Since we are, as of today, unable to prove non-trivial lower bounds for general arithmetic circuits, the emphasis will be to prove lower bounds for restricted classes of circuits. Thus we have the following definitions of certain restricted classes of circuits.

Definition 2.7. *A constant depth circuit of depth* d *is an arithmetic circuit where any path from the output to the input has atmost* d *gates. In particular, a depth-3 circuit is an arithmetic circuit, which has 3 levels of gates. Two special cases of depth-3 circuits are the* $\Sigma\Pi\Sigma$*(which has a* $+$ *gate at the output level, with* $\times$ *in the next and* $+$ *at the lowest level) and* $\Pi\Sigma\Pi$ *circuits.*

Definition 2.8. *A polynomial* $p \in \mathbb{F}[x_1, \ldots, x_n]$ *is* homogeneous *if the degree of each monomial in* p *is the same. A circuit is* homogeneous *if the polynomial computed at each gate in the circuit is homogeneous. A polynomial* $p \in \mathbb{F}[x_1, \ldots, x_n]$ *is* multilinear *if the degree of each variable in a monomial is atmost 1. A circuit is* multilinear *if the polynomials computed at each of the gates is a multilinear polynomial.*

Definition 2.9. *Let the set of variables be partitioned into* d *parts* $X_1, \cdots, X_n$. *A polynomial* p *over these variables is called* set-multilinear *if every monomial in it is of degree* d *and contains exactly one variable from each part. Such a polynomial is both homogeneous and multilinear.*

Note that this is a restriction on multilinear polynomials. For instance the polynomial $p(x_1, x_2, x_3) = x_1x_2 + x_2x_3 + x_3x_1$ is multilinear but not set-multilinear.

2.2 The Partial Derivative Method

In this section, we describe the partial derivative method and its success in proving a few non-trivial lower bounds for some restricted classes of circuits. We measure the complexity of a function as the dimension of the space spanned by the partial derivatives. This is a non-monotone measure in the sense that the dimension can reduce even if the number of monomials increases as we can see in this example.

Example 2.10. *Let* $f_1 = x_1^2 + x_2^2$ *and* $f_2 = x_1^2 + x_2^2 + 2x_1x_2$. *For* f_1*, we have* $\partial(f_1) = \{x_1^2 + x_2^2, 2x_1, 2x_2, 1\}$ *and* $\partial(f_2) = \{x_1^2 + x_2^2 + 2x_1x_2, 2(x_1 + x_2), 2\}$.

In the next section, we survey an old result by Baur and Strassen[BS83], which is the first application of partial derivatives of a polynomial to prove size lower bounds. Then we consider the general technique that is used by both [Nis91] and [NW97] to prove lower bounds for a polynomial p is by obtaining from the structure of the circuit C, of size s, some bound on the dimension of the space spanned by the partial derivatives of any function computed by it. Now, $\dim(\partial(p)) < f(n)$ for some function f, which gives us a lower bound on s. The application of this method on explicit polynomials will also be surveyed in this chapter.

2.2.1 Lower Bound of Baur and Strassen

The approach in [BS83] is to count the number of partial derivatives of a polynomial to obtain an $\Omega(n \log n)$ lower bound for the polynomial

$$f(x_1, \ldots, x_n, y_1, \ldots, y_n) = \sum_{i=1}^{n} y_i x_i^n$$

This comes from an upper bound on the size of the circuit computing the various partial derivatives of f, and a lower bound by Strassen[Str73] that any circuit computing set of polynomials $x_1^n, \dots, x_n^n$ requires $\Omega(n \log n)$ many gates. First, we prove the following theorem.

Theorem 2.11 ([Str73]). *Any arithmetic circuit C that computes all the polynomials $x_1^n, \dots, x_n^n$ requires $\Omega(n \log n)$ gates.*

Before we proceed to the proof of the theorem, we will state a result from algebraic geometry that will be used crucially in the proof.

Theorem 2.12 (Bézout's Theorem). *Suppose we have k polynomials $f_1, \dots, f_k$ over an algebraically closed field $\mathbb{F}$..Let degree of f_i be r_i for $i \in [k]$. Let $S \subseteq \mathbb{F}^n$ be the set of common zeros of these polynomials. Then S is either infinite or $|S| \leqslant \prod_{i=1}^{k} r_i$.*

Consider a set of polynomials equations $\mathcal{P}$

$$\begin{aligned} f_1(x_1, \dots, x_n) &= b_1 \\ f_2(x_1, \dots, x_n) &= b_2 \\ &\vdots \\ f_n(x_1, \dots, x_n) &= b_n \end{aligned}$$

such that the number of common zeroes in $\mathcal{P}$ is n^n.

Lemma 2.13. *Any arithmetic circuit with $+, \times$ gates that computes $f_1, \dots, f_n$ requires $\Omega(n \log n)$ gates.*

Before we prove the lemma, let us see how this implies Strassen's result. Consider the set of polynomials equations

$$\begin{aligned} x_1^n &= 1 \\ x_2^n &= 1 \\ &\vdots \\ x_n^n &= 1 \end{aligned}$$

This has n^n many solutions over complex numbers, and they are the n^{th} roots of unity. Thus the previous lemma implies Theorem 2.11.

Proof. (Proof of Lemma 2.13)
Suppose a circuit C with $+, \times$ gates computes $f_1, \ldots, f_n$. We will construct a set of equations over $x_1, \ldots, x_n$ and some new variables such that there is a one-one correspondence between the solutions of $\mathcal{P}$ and the solutions of these new set of equations.This is done as follows. For every gate c in the circuit, define a variable y_c. Now for each gate in the circuit, including input gates and constants define the following equations $\mathcal{P}'$:

- If c corresponds to a constant value a, add an equation $y_c = a$.
- If c corresponds to an input gate x_i, add an equation $y_c = x_i$.
- If c corresponds to a $+$ gate, with inputs y_{c_1} and y_{c_2}, add an equation $y_c = y_{c_1} + y_{c_2}$.
- If c corresponds to a $\times$ gate, with inputs y_{c_1} and y_{c_2}, add an equation $y_c = y_{c_1} \times y_{c_2}$.
- If c corresponds to the output gate that computes the polynomial f_i, add an equation $y_c = b_i$.

The construction of the set of equations $\mathcal{P}'$ is such that any solution of $\mathcal{P}$ is a solution of $\mathcal{P}'$. Similarly, if we look at the part of the circuit that computes f_i, and any solution of $\mathcal{P}'$ restricted to those equations, they satisfy $f_i = b_i$. Thus $\mathcal{P}'$ has n^n many solutions. Now Bézout's theorem tells us that the number of common zeroes is atmost the product of degrees of each of the polynomial.In $\mathcal{P}'$, except $\times$ gates every other gate has an equation of degree 1, whereas $\times$ gate has an equation of degree 2. Suppose that there m many

$\times$ gates, then we have

$$n^n \leqslant 2^m \text{and hence}$$
$$m \geqslant n \log n$$

Since the size of the circuit $s \geqslant m$, we have $s = \Omega(n \log n)$. $\square$

We can now prove the lower bound of [BS83] which remains the only super-linear lower bound known for general arithmetic circuits.

Theorem 2.14. *Any arithmetic circuit C computing the polynomial*

$$p(x_1, \dots, x_n, y_1, \dots, y_n) = \sum_{i=1}^{n} y_i x_i^n$$

has size $\Omega(n \log n)$.

This lower bound comes from an upper bound of a circuit computing a polynomial and its partial derivatives.

Theorem 2.15 ([BS83])**.** *Let $f(x_1, \dots, x_n)$ be a polynomial computed by an arithmetic circuit of size s with $+, \times$ gates, then there is a circuit of size atmost $5s$ which computes the set of polynomials*

$$\mathcal{P}_f = \left\{ f, \frac{\partial f}{\partial x_i}, \dots, \frac{\partial f}{\partial x_n} \right\}$$

It can be seen that $\{x_1^n, \dots, x_n^n\} \subseteq \mathcal{P}_p$. The fact that any circuit computing the polynomials $\{x_1^n, \dots, x_n^n\}$ requires $\Omega(n \log n)$ and the upper bound of the previous theorem gives the lower bound.

Proof. (Proof of 2.15)
The proof will proceed by induction on the size of the circuit.
Consider a circuit C of size s computing f. Look at a gate c at the maximum

depth. It has two leaves as its children and the function computed at c, f_c is either $x_i + x_j$, $x_i \times x_j$, $x_i + a$, or $x_i \times a$ where a is a constant. Replacing the gate c by a variable y_c, we have a new function computed by the circuit $h(x_1, \ldots, x_n, y_c)$ such that

$$f(x_1, \ldots, x_n) = h(x_1, \ldots, x_n, f_c)$$

Also , we have a circuit C' of size $s-1$ computing h since we have replaced two leaves by one new variable. Hence there is a circuit C'' of size atmost $5(s-1)$ computing h and all its partial derivatives.

With the partial derivatives of h already in our hand , we want to get hold of the partial derivatives of f using an extra 5 gates. From the construction of h, we can see that for any $l \neq i, j$, $\frac{\partial f}{\partial x_l} = \frac{\partial h}{\partial x_l}$ which has already been computed by C''.

By the chain rule of partial derivatives, we have

$$\frac{\partial f}{\partial x_l} = \frac{\partial h}{\partial x_l} + \frac{\partial h}{\partial f_c} \cdot \frac{\partial f_c}{\partial x_l}$$

When $l = i, j$, we can see that we have already computed $\frac{\partial h}{\partial x_l}$. From the output of $\frac{\partial h}{\partial y_c}$, we can get $\frac{\partial h}{\partial f_c}$ by replacing y_c with f_c. This may increase the number of gates by 1. Also $\frac{\partial f_c}{\partial x_l}$ is either a constant or a variable and hence no gates are required. This has to be done for both x_i and x_j Thus we require atmost 5 more gates to compute the set $\mathcal{P}$ from the set of partial derivatives of h.

This proof can also be extended to show that if the circuit computing f has depth d, then the circuit computing $\mathcal{P}_f$ has depth $3d$ $\square$

2.2.2 Lower Bounds for Non-commutative Formulas

In this section, we look at polynomials over the non-commutative ring of polynomials $\mathbb{F}\langle x_1, \cdots, x_n\rangle$. Any polynomial p in this ring is a sum of monomials where each monomial is string over the alphabet $\{x_1, \cdots, x_n\}$. We consider circuits and ABPs where the operations are done over the non-commutative ring. The idea of partial derivatives was used in the paper by [Nis91] to prove exponential lower bounds for the determinant and permanent in non-commutative ABPs.Though the paper doesn't talk of partial derivatives, we can recast the proofs,following the methods of Raz[Raz05], using partial derivatives, where we use the Definition 2.4. The paper not only proves that the size of the ABP is bounded below by the $\sum_d \dim(\partial_d(f))$, but also shows that there exists an ABP of that size. The lower bound is obtained by showing that for permanent and the determinant, $\dim(\partial_d(f)) = \binom{n}{d}$ where non-commutative determinant and permanent is defined as follows

$$\mathrm{Det}(x_{1,1}, \ldots, x_{n,n}) = \sum_{\sigma \in S_n} \mathrm{sgn}(\sigma) \prod_i x_{i,\sigma(i)}$$

$$\mathrm{Perm}(x_{1,1}, \ldots, x_{n,n}) = \sum_{\sigma \in S_n} \prod_i x_{i,\sigma(i)}$$

over the non-commuting variables $x_{1,1}, \ldots, x_{n,n}$. This also gives a lower bound for the non-commutative formula size of permanent and determinant.The theorem we want to prove is

Theorem 2.16 ([Nis91]). *Any non-commutative formula that computes the determinant and permanent polynomial requires $2^{\Omega(n)}$ many gates*

We will prove this in many stages. First we will show that non-commutative formulas can be converted to homogeneous ABPs without a large blow-up in size. Then we can prove that for any function f computed by a homogeneous ABP of size s, the dimension of the space spanned by the set of all partial derivatives of f, ie $\dim(\partial(f)) \leqslant s$. We will complete the proof by showing that $\dim(\partial(\mathrm{perm}(X)))$ and $\dim(\partial(\det(X))) \geqslant 2^n$.

Lemma 2.17. *Let $p(x_1, \ldots, x_n)$ be a polynomial of degree d computed by a non-commutative formula F of size s. Then there exists $d+1$ non-commutative ABPs such that the i^{th} ABP computes the part of p of degree $i-1$, and the size of each ABP is atmost $d(s+1)$.*

Proof. First we will construct an ABP A from the formula F where the edge labels of A may be affine forms. This conversion is done recursively. Let c be a gate in the formula F:

- If c is an input gate with variable x_i, construct an ABP with just a source s and sink t and join s and t with an edge labelled with x_i.

- If c is a gate that corresponds to a constant a, construct an ABP again with just a source s and sink t and connect them with an edge labelled a.

- If c is a $+$ gate with children c_1 and c_2. Let A_{c_1} and A_{c_2} be the ABPs corresponding to c_1 and c_2. The ABP corresponding to c will have a source s and sink t, with s connected to the source of A_{c_1}, the sink of A_{c_1} connected to the source of A_{c_2} and the sink of A_{c_2} connected to t.

- If c is a $\times$ gate with children c_1 and c_2. Let A_{c_1} and A_{c_2} be the ABPs corresponding to c_1 and c_2. The ABP corresponding to c will have a source s and sink t, with s connected to the source of A_{c_1}, and the source of A_{c_2}. The sink of both A_{c_1} and A_{c_2} is connected to t.

Now we will convert A into an ABP, as in the definition of [Nis91]. This procedure will give us a set of ABPs where the edges are labelled by linear forms. For each node v in A except the source and the sink, split v into $d+1$ many nodes $(v,0),(v,1),\ldots,(v,d)$. (v,i) will compute the degree i homogeneous part of the polynomial computed at v. If there is an edge labelled x_i from u to v in A, connect (u,i) to $(v,i+1)$ for all $i \in [d]$ in the new ABP. Similarly if the edge from u to v is labelled by a constant in A, then in the new ABP there are edges between (u,i) and (v,i) labelled by the constant. Now we delete all the edges (u,v) labelled by the constants, and for every u' which has an edge into u labelled by l, we add an edge from u' to v labelled $a.l$, where a was the constant labelling the edge from u to v. This new ABP will have many sinks, each corresponding to a homogeneous part. So we take as many copies as there are homogeneous part and assign the sinks accordingly.

Notice that the new ABPs that we constructed are layered. This means that the nodes of the ABP can be partitioned to mutually disjoint layers from 1 to d and every edge goes from layer i to $i+1$. □

From now onwards we will assume that our ABPs are layered and that edges are labelled by linear forms. We will now connect the size of such an ABP with the dimension of the space of partial derivatives.

Lemma 2.18 ([Nis91]). *Let $f(x_1,\ldots,x_n) \in \mathbb{F}\langle x_1,\ldots,x_n\rangle$ be a polynomial computed by an ABP A, with source s and sink t of size S. Then $\dim(\partial(f)) \leqslant S$.*

Proof. Notice that for any two nodes u,v in A, we can look at the polynomial $f_{u,v}$ where

$$f_{u,v} = \sum_{\text{all } u-v \text{ paths } p} \prod_{i=1}^{k} l_i^p(x_1,\ldots,x_n)$$

where $l_1^p, \ldots, l_k^p$ are the linear forms along the path p. Thus $f(x_1, \ldots, x_n) = f_{s,t}$. Let $v_1, \ldots, v_k$ be the nodes in the first layer of A and the edges (s, v_i) are labelled by the linear forms $l_{s,i}^1$. Then we have

$$f(x_1, \ldots, x_n) = \sum_{i=1}^{k} l_{s,i}^1 \cdot f_{i,t}$$

Thus for any $i \in [k]$, we have

$$\frac{\partial f}{\partial x_i} = \sum_{i=1}^{k} c_i \cdot f_{i,t} \text{ where } c_i\text{s are constants}$$

Similarly, when we look at nodes $v_1, \ldots, v_k$ in some layer m, we can write

$$\begin{aligned} f(x_1, \ldots, x_n) &= \sum_{i=1}^{k} f_{s,i} \cdot f_{i,t} \text{ and} \\ \frac{\partial^m f}{\partial x_{i_1} \ldots \partial x_{i_m}} &= \sum_{i=1}^{k} c_i \cdot f_{i,t} \text{ where } c_i\text{s are constants} \end{aligned}$$

Thus any partial derivative of f can be written as a linear combination of $f_{u,t}$ and since there are S nodes in the ABP, $\dim(\partial(f)) \leqslant S$. $\square$

Now to complete the proof of the main theorem in this section, we have the following lemma[Nis91].

Lemma 2.19. *Let* $\mathrm{Perm}(x_{11}, \ldots, x_{nn})$ *denote the permanent polynomial.*

$$\dim(\partial(\mathrm{Perm})) \geqslant 2^n$$

.

Proof. $\mathrm{Perm}(\cdot)$ is a multilinear polynomial of degree n. Thus any non-zero partial derivative of it is of order atmost n. For any $d \leqslant n$, choose d indices $i_1, \ldots, i_d$ and look at the polynomial $\frac{\partial(\mathrm{Perm})}{\partial x_{1,i_1}, \ldots, x_{d,i_d}}$. This amounts to choosing the first d rows and columns $i_1, \ldots, i_d$ and looking at the permanent submatrix obtained after the removal of these rows and columns. For distinct $i_1, \ldots, i_d$, these polynomials are linearly independent since each will have atleast one new monomial. Similarly, when we look at derivatives of order d_1 and order d_2 with $d_1 \neq d_2$, then the resultant polynomials are of degree $n - d_1$ and $n - d_2$. Thus these are also linearly independent. Thus we have atleast

$$\sum_{d=0}^{n} \binom{n}{d} = 2^n$$

linearly independent partial derivatives. Thus we have $\dim(\partial(\mathrm{Perm})) \geqslant 2^n$ which proves the lemma and consequently the theorem. □

Note that this same argument goes through if we replace the permanent by the determinant. The determinant polynomial is the same as permanent except that we now look at the sign of each permutation. This does not affect the fact that the different partial derivatives are linearly independent and hence we get the same lower bound for the determinant polynomial also.

2.2.3 Lower Bounds for Commutative Circuits

In this section, we will survey the application of the method of bounding the dimension of the space of partial derivatives to obtain lower bounds in the case of certain restricted classes of commutative circuits[NW97]. Two explicit polynomials against which we will prove lower bounds are the symmetric polynomials and the iterated matrix multiplication.

The d^{th} symmetric polynomial over n variables is defined thus

$$\mathrm{Sym}_n^d(x_1, \ldots, x_n) = \sum_{S \subseteq [n], |S| = d} \prod_{i \in S} x_i$$

Non-trivial lower bounds in the case of symmetric polynomials are not known for general circuits. The first result which shed some light on the problem was [NW97], although the result was for homogeneous depth 3 circuits over fields of characteristic zero. Shpilka and Wigderson[SW01] improved this to prove the following result about symmetric polynomials for general depth 3 circuits again over fields of characteristic zero. Good lower bounds for the symmetric polynomial over finite fields are not known.

Theorem 2.20 ([SW01]). *For every* $\log n \leqslant d \leqslant \frac{n}{10}$*, size of depth 3 circuit computing* Sym_n^{2d} *is* $\Omega(\frac{n^2}{d})$.

This was improved by Shpilka[Shp02] to the following

Theorem 2.21. *For every* $d = \alpha n$*, where* $0 < \alpha < 1$ *is a constant, the size of the depth 3 circuit computing* Sym_n^d *is* $\Omega(n^2)$.

This matches with the following upper bound given by Ben-Or for symmetric polynomials computed by depth-3 circuits.

Theorem 2.22 (Ben-Or). *For every* d*, there exists* $O(n^2)$ *sized depth-3 circuits computing symmetric polynomials of degree* d *over fields of characteristic zero.*

Proof sketch. The d^{th} symmetric polynomial $\mathrm{Sym}_n^d(x_1, \ldots, x_n)$ is the coefficient of t^{n-d} in the polynomial $\prod_{i=1}^{n}(t+x_i)$. We have an $O(n)$ sized $\Pi\Sigma$ circuit that computes this polynomial. Since this polynomial contains all the symmetric polynomials and we require only the d^{th} one, we will interpolate this polynomial at $n+1$ values for t to obtain the correct symmetric polynomial. This gives us the $O(n^2)$ depth 3 circuit for the symmetric polynomials. $\square$

We will now prove the following theorem from [NW97] which says that the homogenous restriction on the circuit is rather severe. In this scenario we have an exponential lower bound compared to the polynomial sized circuit for general depth 3 circuits.

Theorem 2.23. *Any homogeneous depth 3 circuit computing* Sym_n^{2d} *on* n *variables over a field of characteristic zero requires* $\Omega\left(\left(\frac{n}{4d}\right)^d\right)$ *gates.*

For proving the theorem, we first observe the following properties about the dimension of the space spanned by the partial derivatives.

Observation 2.24. *Let* $f_1, \ldots, f_k \in \mathbb{F}[x_1, \ldots, x_n]$ *and let* $\alpha \in \mathbb{F}$.

- $\dim(\partial(\alpha f_i)) = \dim(\partial(f_i))$
- $\dim(\partial(\sum_{i=1}^k f_i)) \leqslant \sum_{i=1}^k \dim(\partial(f_i))$
- $\dim(\partial(\prod_{i=1}^k f_i)) \leqslant \prod_{i=1}^k \dim(\partial(f_i))$

When we consider depth 3 homogeneous circuit, we can assume the circuit to be $\Sigma\Pi\Sigma$. Suppose instead we have a $\Pi\Sigma\Pi$ circuit computing Sym_n^d, the topmost $\times$ gate will have fan-in atmost d since we are computing degree d polynomial and the circuit is homogeneous. Since Sym_n^{2d} is a multilinear polynomial, the $+$ gates at level two will be multilinear. This would give us lower bound of $\binom{n}{2d}$ for a $\Pi\Sigma\Pi$ circuit computing Sym_n^{2d}. So we will look at the more interesting case of $\Sigma\Pi\Sigma$ circuits. We have the following lemma

Lemma 2.25 ([NW97]). *Let* f *be computed by a depth 3 circuit with fan-in* s *at the top* $+$ *gate and fan-in atmost* d *at every* $\times$ *gate. Then* $\dim(\partial(f)) \leqslant s \cdot 2^d$.

Proof. For every linear function l computed at the bottom $+$ gate, $\dim(\partial(l))$ $= 2$. Since the fan-in of every $\times$ gate is atmost d, for any function g computed by a multiplication gate $\dim(\partial(g)) \geqslant 2^d$ by the earlier observation. The lemma follows from the fact that the topmost $+$ gate has fan-in s and the earlier observation. $\square$

Suppose we manage to prove that $\dim(\partial(Sym_n^{2d})) \geqslant \Omega((\frac{n}{2d})^d)$, then we would have proved the theorem of [NW97]. Thus to complete the proof, we have the following lemma

Lemma 2.26.

$$\dim(\partial(Sym_n^{2d}(x_1, \dots, x_n))) \geqslant \binom{n}{d}$$

Proof. We will prove the lemma by showing that a subset of the set of all partial derivatives, spans a vector space of high dimension. For proving this let us look at the set U and V defined as follows.

$$U = \left\{ \prod_{i \in S} x_i : S \subseteq [n], |S| = d \right\}$$
$$V = \left\{ \frac{\partial(Sym_n^{2d})}{\partial x_{i_1} \dots \partial x_{i_d}} : i_1 < \dots < i_d, 1 \leqslant i_1, \dots, i_d \leqslant n \right\}$$

Notice that $V \subseteq \partial(Sym_n^{2d})$.

We can think of the U and V as vectors indexed by subsets of n of size d in lexicographic order. Differentiating Sym_n^{2d} with respect to a set of variables $\{x_{i_1}, \dots, x_{i_d}\}$ is equivalent to looking at the symmetric polynomial Sym_{n-d}^{d} where we don't look at the variables $x_{i_1}, \dots, x_{i_d}$.Thus we can write $V = DU$, where D is an $\binom{n}{d} \times \binom{n}{d}$ indexed by d sized subsets of n.

$$D_{S,T} = \begin{cases} 1 \text{ if } S \cap T = \phi \\ 0 \text{ otherwise} \end{cases}$$

where S and T are d sized subsets of $[n]$. D is known as the disjointness matrix

It is known that the disjointness matrix is of full rank over fields of characteristic zero[Got66]. Thus $\dim(V) = \dim(U)$. Now $\dim(\partial(Sym_n^{2d})) \geqslant \dim(V)$. Since each polynomial is infact just a monomial and they have different variables means that $\dim(U) = \binom{n}{d}$. Thus we have the lemma. □

We can use the technique of partial derivatives to obtain lower bounds for the iterated matrix multiplication polynomial for depth 3 homogeneous and set-multilinear circuits. The iterated matrix multiplication(IMM_d^n) is to compute the $(1,1)^{th}$ entry of the matrix formed by the product of d $n \times n$ matrices. Since the polynomial itself is a set-multilinear polynomial(where the variable set is divided into d parts with each part containing n^2 variables), it is interesting to look at the lower bound for set-multilinear circuits rather that general multilinear circuits. To bound the dimension of the space spanned by the partial derivatives of IMM_d^n, we have

Lemma 2.27. *For every* d, n, $\dim(\partial \mathrm{IMM}_d^n) \geqslant n^{d-1}$

Proof. Note that IMM_d^n is a polynomial that depends on all the entries of all the matrices except the first and the last matrices. It depends only on the first row of the first matrix and the first column of the last matrix. Consider the set of partial derivatives obtained by differentiating with respect to the variables in even numbered matrices. Since each of these polynomials have monomials which depend on different variables, they are all linearly independent.

We can show that there are n^{d-1} such polynomials. Suppose d is odd, then there are $\frac{d-1}{2}$ matrices each containing n^2 variables giving us n^{d-1} polynomials. If d is even, then we can take partial derivatives w.r.t to n^2 variables in the $\frac{d}{2} - 1$ matrices and n variables in the last matrix. Thus we have n^2 many polynomials.This gives the lower bound for the dimension of the space spanned by the partial derivatives. □

When we are working with the set-multilinear polynomials, we use a slightly different definition for the dimension of the space spanned by its partial derivatives. This definition will be useful in getting the lower bounds for the size of set-multilinear circuits, even though they don't measure the actual dimension of the space spanned by the partial derivatives.

Let f be a set-multilinear polynomial over the variables $X = \dot{\bigcup}_{i=1}^{d} X^i$, where each of the X^is contain n^2 variables $x_1^i, \cdots, x_{n^2}^i$. For a $T \subseteq [d]$, P_T denotes the set-multilinear polynomials over the set T. In particular, the monomials of P_T will be a product of variables with one variable taken from each set of T. Thus we have $\dim(P_T) = n^{2|T|}$. Denote by $\partial_S(f)$ the partial derivatives of f with respect to the monomials in P_S. By $\dim_{SM}(f)$, we denote $\max_{S \subseteq T} \dim(\partial_S(f))$. Notice that according to this definition, we have the following lemma similar to 2.27

Lemma 2.28. *For every d, n, we have $\dim_{SM}(IMM_d^n) = n^{d-1}$.*

With this in hand, we can prove exponential lower bounds for the size of set-mulitlinear depth 3 $\Sigma\Pi\Sigma$ circuits computing IMM_d^n. First note that we can convert $\Sigma\Pi\Sigma$ homogenous circuits for a set-multilinear polynomial f into $\Sigma\Pi\Sigma$ set-multilinear circuits.

Lemma 2.29. *For every set-multilinear polynomial f on $X = \dot{\bigcup}_{i=1}^{d} X^i$, size of set-multilinear $\Sigma\Pi\Sigma$ circuits for f is atmost $d!$ times the size of homogeneous $\Sigma\Pi\Sigma$ circuits for f.*

Proof-Sketch. Given a homogeneous $\Sigma\Pi\Sigma$ circuit for f, we will first replace each $+$ gate at the bottom level with d gates where each one adds up one part of the partition of X.Now for each multiplication gate, replace it by $d!$ gates with each gate choosing one part from the bottom d addition gates. Now the final addition gate adds up all the multiplication gates that are used for the function f and since f is set-multilinear this procedure works. $\square$

Theorem 2.30 ([NW97]). *For all d and n, $\text{size}_3^{SM}(IMM_d^n) \geqslant n^{d-1}$, and $\text{size}_3^{H}(IMM_d^n) \geqslant \frac{n^{d-1}}{d!}$.*

Proof. For a set-multilinear $\Sigma\Pi\Sigma$ circuit, the bottom $+$ gates computes a polynomial of the form $p(X^j) = \sum_{i=1}^{n^2} x_i^j$. Thus $\dim_{SM}(p) = 1$. Hence the dimension of the $\times$ gate is 1. But we know from lemma 2.28 that $\dim_{SM}(IMM_d^n) = n^{d-1}$, Thus $size_3^{SM}(IMM_d^n) \geqslant n^{d-1}$ and from lemma 2.29, we get $size_3^{H}(IMM_d^n) \geqslant \frac{n^{d-1}}{d!}$. $\square$

In the case of general set-multilinear circuits, [NW97] prove an exponential lower bound for the IMM_d^2 polynomial.

Theorem 2.31. *For any* d, h, $size_h^{SM}(IMM_d^2) = 2^{\Omega(d^{\frac{1}{h}})}$.

The crucial part in the proof is defining a new measure of complexity which in some sense captures how close is the dimension of the partial derivative space to the maximum that is achievable. The idea in [NW97] approach is to show that even after assigning a random restriction, this measure doest not drop for the polynomial IMM_d^2, whereas for any small set-multilinear circuit this is not true. To formalize this idea into a proof, we need some definitions and observations.

Observation 2.32. *For* $T \subseteq [d]$, $\dim(P_T) = n^{2|T|}$.

Observation 2.33. *Let* $f \in P_T$, *then* $\dim(\partial_S(f)) \leqslant \min\{n^{2|S|}, n^{2|T-S|}\} \leqslant n^{2\lfloor\frac{|T|}{2}\rfloor}$.

For convenience, we will follow [NW97] notation and denote $2\lfloor\frac{|T|}{2}\rfloor$ by $\langle|T|\rangle$. Thus based on observation, we can define the measure $\rho(f) = \frac{\dim_{SM}(f)}{n^{\langle|T|\rangle}}$, where $f \in P_T$. Thus $\rho(f) \leqslant 1$. From the definition of ρ we have the following observations

Observation 2.34. *Consider a set-multilinear circuit* C.

1. *For a* $+$ *gate computing* f *where the inputs are the polynomials* $g_1, \dots, g_s$, $\rho(f) \leqslant \sum_{i=1}^{s} \rho(g_i)$.

2. *For a* $\times$ *gate computing* f *where the inputs are the polynomials* $g_1, \dots, g_s$, *where* m *many of them are odd degree polynomials, then* $\rho(f) \leqslant \prod_{i=1}^{s} \rho(g_i) \leqslant n^{-\langle m\rangle}$

We will need a couple of lemmas and the notion of a random restriction before we can prove theorem 2.31.

Lemma 2.35 ([NW97]). *Let C be an optimal set-multilinear circuit computing a polynomial $f \in P_[d]$, where the multiplication depth is h. Then there are $\times$ gates $g_1, \ldots, g_s$ in the circuit with the following properties.*

1. $s \leqslant size_h^{SM}(f)$.
2. *For every $j \in [s]$, the fan-in of g_i is atleast $d^{\frac{1}{h}}$.*
3. $\sum_{i=1}^{s} \rho(g_i) \geqslant \rho(f)$.

Proof. If the topmost gate of the circuit is a $\times$ gate, then we are done, since that gate itself satisfies all the three conditions. Assume that the topmost gate is a $+$ gate. Then along each path from it to an input node, find the first $\times$ gate that satisfies condition 2. Since the polynomial computed has degree d and the multiplication depth is h, there will exist atleast one such gate in every path. Call these $g_1, \ldots, g_s$. These definitely satisfy condition 1. Notice that the function depends on all these $\times$ gates since we considered all paths from the output node to the input nodes. Thus condition 3 is satisfies because of the previous observation. □

The following lemma follows from Chernoff bounds[HR90]

Lemma 2.36. *Let $z_1, \ldots, z_d$ be independent 0,1 random variables. For $T \subseteq [d]$, let $z(T) = \sum_{i \in T} z_i (mod\ 2)$.*

1. $\Pr[\sum_i z_i < \frac{d}{3}] \leqslant \frac{1}{10}$, *for* $d \geqslant 20$.
2. *For every family of pairwise disjoint* $T_1, \ldots, T_k \subseteq [d]$, $\Pr[\sum_i z(T_i) < \frac{r}{3}] \leqslant 2^{-\frac{r}{10}}$.

Now let us define a random restriction. Let $f \in P_{[d]}$ be set-multilinear polynomial. A random restriction is a set $R \subseteq [d]$, where we assign values to all variables in $X^i, i \in R$. The crucial fact about small-sized set-multilinear circuits that we will use is that on most random restrictions, the dimension of its partial derivative space drops by a large amount.

Lemma 2.37 ([NW97]). *Let d, h be integers and let $r = d^{\frac{1}{h}} \geqslant 20$. Let $z = (z_1, \ldots, z_d)$ be a sequence of independent random variables. Let $Z \subseteq [d]$ denote the positions that we set to 1 in z(This will denote the random restriction). Let $f \in P_{[d]}$ be such that $\text{size}_h^{SM}(f) \leqslant \frac{1}{2} \cdot 2^{\frac{r}{10}}$. Then $\Pr[\rho(f\big|_Z) \geqslant \frac{1}{n}] \leqslant \frac{1}{2}$.*

2.31 follows from this lemma since, after a random restriction of the polynomial IMM_d^2, we are left with IMM_t^2 for some $t < d$. Thus $\Pr[\rho(f\big|_Z) \geqslant \frac{1}{n}] = 1$, which implies $\text{size}_h^{SM}(IMM_d^2) = 2^{\Omega(d^{\frac{1}{h}})}$.

We reiterate that the results stated here are not exhaustive but merely representative of a technique which has led to some non-trivial results. The current techniques don't seem to lead to good size lower bounds for even constant depth circuits let alone general arithmetic circuits.

Chapter 3

The Partial Derivative method in Learning Algorithms

3.1 Preliminaries

In this chapter we switch to a different problem and look at algorithms for *learning* arithmetic circuits. The reason for our interest is that similar methods based on partial derivatives succeed in giving efficient learning algorithms. The model of learning that we will be interested in will be the *exact learning* model of Angluin[Ang87]. In this model of learning, we have a learner who has access to the function f(which is the target function to be learned) as a black-box. The learner is allowed two types of queries to learn the function:

1. *Membership queries*:The learner chooses an input x in the domain of the function and queries the black-box for the value $f(x)$.

2. *Equivalence queries*:The learner constructs a hypothesis function h and queries the black-box whether this is actual function f. If the two are different, then the black-box returns a *counter-example* x in the domain such that $f(x) \neq h(x)$.

We say that the class of functions $\mathcal{C}$ is *efficiently learnable*, if for all function $f \in \mathcal{C}$ there is an algorithm that runs in time polynomial in the size of the representation of f and the longest counter-example and outputs a hypothesis h that is equivalent to f.

This is different from the *Probably Approximately Correct*(PAC) model of Valiant[Val84], where the learner is allowed to look at a set of pairs $(x, f(x))$ picked from some distribution. Since the number of zeroes of a polynomial is atmost its degree, learnability in the *exact learning* model implies learnability in the PAC model, but now with membership queries[Val84, Ang87].

We define a few notions that will be useful in this chapter.

Definition 3.1 (Multiplicity Automaton,[KS06]). *Fix an alphabet Σ with $s = |\Sigma|$ elements. A multiplicity automaton A of size r over Σ is a vector $\hat{\gamma} = (\gamma_1, \ldots, \gamma_r)$ and a set of s matrices μ_σ for each $\sigma \in \Sigma$, where μ_σ is an $r \times r$ matrix whose elements come from a field $\mathbb{F}$. The output of A on an input $w = w_1 \ldots w_n$ is defined as $(\prod_{i=1}^{n} \mu_{w_i})_1 \cdot \hat{\gamma}^T$ where $(\prod_{i=1}^{n} \mu_{w_i})_1$ is the first row of $(\prod_{i=1}^{n} \mu_{x_i})$. For a string $w = w_1 \ldots w_n$, we define $\mu_w = \prod_{i=1}^{n} \mu_{w_i}$.*

A multiplicity automaton $\mathcal{A}$ can be thought of as a generalization of the ordinary finite state automaton with the following additions. $\mathcal{A}$ is an automaton with r states and the matrices μ_σ give the labels of the transitions given by the symbol $\sigma \in \Sigma$. Thus on an input x, the automaton multiplies the labels corresponding to each transition given by the input and also the weight of the final state given by $\hat{\gamma}$ and sums it over all the final states. When the matrices corresponding to symbols from a string is w, the $(i, j)^{th}$ entry of the product gives the weighted sum of all paths labelled w from i to j.

We can understand this better with an example

Example 3.2 ([BR88]). *Consider the function* $f : \Sigma \to \mathbb{R}$*, over* $\Sigma = \{a, b\}$ *where* f *computes the number of* a*'s in a string* $w \in \Sigma^*$*. We have a two state multiplicity automaton where*

$$\mu_a = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$$
$$\mu_b = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

and $\hat{\gamma} = (0, 1)$*. Since* μ_b *is the identity matrix, the number of* a*'s in the string* w *is given by the first row of* $\mu_a^k \cdot \hat{\gamma}$*, where* k *is the number of* a*'s in* w*. But*

$$\mu_a^k = \begin{pmatrix} 1 & k \\ 0 & 1 \end{pmatrix}$$

and we get $\mu_w \cdot \hat{\gamma} = k$*.*

Now, we define the Hankel matrix

Definition 3.3 ([BBB$^+$00]). *Consider a function* $f : \Sigma^n \to \mathbb{F}$ *where* Σ *is some alphabet and* $\mathbb{F}$ *is a field. The Hankel matrix corresponding to* f*,* H *will be indexed by strings from* Σ^* *of length atmost* n*. For two strings* $x \in \Sigma^d$ *and* $y \in \Sigma^{n-d}$*, the* $(x, y)^{th}$ *entry of* H *is* $f(x \cdot y)$ *where '$\cdot$' is concatenation. For strings* x, y *such that* $x + y \neq n$*, the* $(x, y)^{th}$ *entry is zero. We will also look at* H_k *which is the submatrix of* H *indexed by strings of length* k *and* $n - k$*.*

In this chapter, we first show that multiplicity automata as defined above are efficiently learnable. This is due to [BBB$^+$00], who extend the theorem of [CP71, Fli74] which relates the size of the smallest multiplicity automata computing a function f to the rank of its Hankel matrix, to an algorithm which learns the multiplicity automata. Then we go on to show that arithmetic circuits, whose space of partial derivatives is of small dimension, has a small multiplicity automata and thus can be learned efficiently[KS06].

3.2 Learnability of Multiplicity Automata

In this section we look at the result due to [BBB⁺00] that functions that have small rank Hankel matrix, has a small sized multiplicity automaton which can be computed efficiently. We will prove a theorem due to [CP71, Fli74] which connects the size of the minimal automaton computing a function f and the rank of the Hankel matrix. The same proof is also used in the algorithm to learn a multiplicity automaton.

Theorem 3.4. *Let $f : \Sigma^n \to \mathbb{F}$ be a function that is not identically zero and let H_f be the Hankel matrix corresponding to f. Then the size of the smallest multiplicity automaton computing f is the same as the rank of H_f.*

Proof. ($\Rightarrow$) Let A be a multiplicity automaton which computes f with s states. Consider the following matrices : P whose rows are indexed by Σ^n and the columns are indexed from 1 to s, and S whose rows are indexed from 1 to s and the columns are indexed by strings of Σ^n and the matrices are defined as follows

$$P(x, i) = \mu_x(1, i), \text{ i.e the } (1, i)^{th} \text{ entry of } \mu_x$$
$$S(i, x) = (\mu_x)_i \cdot \hat{\gamma}, \text{ dot product of } i^{th} \text{ row of } \mu_x \text{ and } \hat{\gamma}$$

Now we can show that $H_f = P \cdot S$. For this we have

$$\begin{aligned} H_f(x \cdot y) &= f(x \cdot y) = (\mu_{x \cdot y})_1 \cdot \hat{\gamma} \\ &= (\mu_x \mu_y)_1 \cdot \hat{\gamma} \\ &= \sum_{j=1}^{s} \Big(\sum \mu_x(1,k)\mu_y(k,j) \Big) \cdot \hat{\gamma}_j \end{aligned}$$

which can be written as $\sum_{j=1}^{s} \mu_x(1,i) \cdot (\mu_y)_i \cdot \hat{\gamma}$, which according to the definition of P and S is the product of these two matrices. Thus $H_f = P \cdot S$.
($\Leftarrow$) Let $x_1, \dots, x_r$ be the index of the rows that are linearly independent in H_f. Now we need to construct a multiplicity automaton A for which we have to define the matrices μ_σ for every $\sigma \in \Sigma$ and the vector $\hat{\gamma}$.We will assume that x_1 is the empty string, since we can construct a set of linearly independent vectors starting from starting from any vector. Define $\hat{\gamma}$ to be the vector $(f(x_1), \dots, f(x_r))$. For each $\sigma \in \Sigma$ define the matrix μ_σ as follows: The i^{th} row of μ_σ will be the coefficient of the rows corresponding to $x_1, \dots, x_r$ when the row $x_i \cdot \sigma$ is written as a linear combination of these rows.i.e

$$(H_f)_{x_i \cdot \sigma} = \sum_{j=1}^{r} \mu_\sigma(i,j)(H_f)_{x_j}$$

Now to show that this multiplicity automaton indeed computes f, we use induction on the length of the string $w \in \sigma^n$. The claim we will prove is as follows

Claim. For any string $w \in \sigma^n$, $(\mu_w)_i \cdot \hat{\gamma} = f(x_i \cdot \gamma)$, for all i.

This claim will imply the theorem, since $(\mu_w)_1 \cdot \hat{\gamma} = f(x_1 \cdot w) = f(w)$. The claim is true when $w = \varepsilon$ since in that case μ_w is the identity matrix and therefore $(\mu_w)_1 \cdot \hat{\gamma} = f(x_i)$ by the definition of $\hat{\gamma}$. Let σw be string of length $n+1$. Then

$$\begin{aligned} f(x_i \cdot \sigma \cdot w) &= H_f(x_i \cdot \sigma, w) \\ &= \sum_{j=1}^{r} \mu_\sigma(i,j) \cdot H_f(x_j, w) \end{aligned}$$

But from the definition of the Hankel matrix, we know $H_f(x_j \cdot w) = f(x_j \cdot w)$. Thus using the induction hypothesis on w, we have

$$\begin{aligned} f(x_i \cdot \sigma \cdot w) &= \sum_{j=1}^{r} \mu_\sigma(i,j) \cdot (\mu_w)_j \cdot \hat{\gamma} \\ &= (\mu_\sigma \cdot \mu_w)_i \cdot \hat{\gamma} = (\mu_{\sigma \cdot w})_i \cdot \hat{\gamma} \end{aligned}$$

which completes the induction and we have the theorem. $\square$

We will now look at a simple example to illustrate this connection between the rank of the Hankel matrix of a function and the size of the multiplicity automaton computing it.

Example 3.5. *Let* $\Sigma = \{x, y\}$ *and consider the function* $f : \Sigma^n \to \{0,1\}$ *defined such that*

$$f(w) = \begin{cases} 1 & \textit{if } w = xx^R, x \in \Sigma^{\frac{n}{2}} \\ 0 & \textit{otherwise} \end{cases}$$

Let H *denote the Hankel matrix of this function. Consider the submatrix* $H_{\frac{n}{2}}$ *which is an* $\frac{n}{2} \times \frac{n}{2}$ *identity matrix(since corresponding to a string* w *there is exactly one string which is the reverse of it). Thus the rank of the Hankel matrix* H *is atleast* $2^{\frac{n}{2}}$*. Thus any multiplicity automaton computing it is of size atleast so much. From this we can also see that there does not exist any finite state automaton accepting the language corresponding to this function.*

We can now look at the algorithm given by [BBB+00] for learning a multiplicity automaton. As mentioned earlier, the learner can query the function for the value at a particular input as well as query whether a hypothesis is correct. If it is incorrect, then the function gives a counter-example. The running time of the algorithm will be polynomial in the size of the automaton and the length of the counter-example produced.

The algorithm computes the linearly independent rows in the Hankel matrix and constructs the automaton as in the proof given earlier. For this it maintains a list of rows and columns and adds more into it, if it finds a new row that is linearly independent with these. From now on we will omit the subscript from H denoting the function as it is implicit. Thus given a set of rows indexed by the strings $x_1, \ldots, x_l$ and columns indexed by $y_1, \ldots, y_l$, by H_{x_i} we denote the row in H corresponding to the string x_i. By $\hat{H}_{x_i}$ we will denote the row corresponding to x_i restricted to the columns $y_1, \ldots, y_l$.

The algorithm is as follows(We will assume here that the function f on input ε computes a non-zero value. If not, we learn the function where it computes a non-zero value and then change the automaton accordingly).

1. $x_1 \leftarrow \varepsilon$, $y_1 \leftarrow \varepsilon$, $X \leftarrow \{x_1\}$, $Y \leftarrow \{y_1\}$, $l \leftarrow 1$

2. Construct a hypothesis h as in the proof of the theorem, by looking at the matrix restricted to X and Y. Here $\hat{\gamma} = (f(x_1), \ldots, f(x_l))$ and μ_σ is obtained by expressing the row $\hat{H}_{x \cdot \sigma}$ as the linear combination of $\hat{H}_{x_1}, \ldots, \hat{H}_{x_l}$.

3. Ask an equivalence query with h. If the black-box answers YES we are done, else let z be the counter-example.

4. If we get a counter-example z, then find a prefix of z, $w \cdot \sigma$ such that

- the row corresponding to w can be written as a linear combination of the rows x_i with the coefficients from the matrix μ_w
- the row $w \cdot \sigma$ cannot be written as a linear combination of the rows in X. Let y be the column where it is different.

Then add w to X, $\sigma \cdot y$ to Y and increment l and continue creating a new hypothesis.

To argue about the correctness of the above algorithm[BBB+00], we need to show that at step 2 the vectors given by X and Y are linearly independent.Secondly, that when a counterexample z is produced we can find a prefix $w \cdot \sigma$ satisfying the conditions in step 4. This can be proved by showing that if no prefix $w \cdot \sigma$ satisfies both the conditions of step 4, then infact $f(z) = h(z)$ which is a contradiction. The fact that adding new vectors to X and Y keeps the vectors linearly independent follows from the fact that the new vectors that we add depend on the prefix which wasn't expressible as a linear combination of the other vectors in X.

3.3 Improper Learning algorithms for Arithmetic Circuits

In this section, we apply the techniques that we developed in the last section to give learning algorithms for certain restricted classes of arithmetic circuits. The algorithms are obtained by observing that the rank of the multiplicity automaton is upper bounded by the dimension of the space spanned by the partial derivatives of the circuit[KS06].

The algorithms presented are improper learning algorithms. This means that the hypothesis produced by the algorithm does not belong to the class of functions learned. More specifically in our case, the class of functions that we learn are restricted arithmetic circuits, whereas the hypothesis produced by the algorithm is a multiplicity automaton computing it. The algorithms run in time polynomial in the size of the field and the rank of the Hankel matrix of the function. Before we get into the details regarding the learning algorithms we have a few definitions

Definition 3.6 (Evaluation Matrix). *Let f be a set-multilinear polynomial over the variable $X = \dot{\bigcup}_{i=1}^{d} X_i$ over a field $\mathbb{F}$. The evaluation matrix E_k is defined as follows:Order the elements of $\mathbb{F}^{d-k}$ in lexicographic order and they index the columns of the matrix E_k. The rows are indexed by the set-multilinear monomials where the variables come from $X_1, \cdots, X_k$. The value at the location of the matrix indexed by the row $x_{1,i_1} \cdots x_{k,i_k}$ and the column $\rho = (x_1, \cdots, x_{d-k})$ is*

$$\frac{\partial f}{\partial x_{1,i_1} \cdots \partial x_{k,i_k}}(\rho)$$

Definition 3.7 (Generalized Vandermonde Matrix). *Let f be a set-multilinear polynomial over the variables $X = \dot{\bigcup}_{i=1}^{d} X_i$ over a field $\mathbb{F}$. The coefficient matrix $V_{k,d}$ has columns indexed by the set-multilinear monomials in $\{X_1, \cdots, X_k\}$ of degree d and the rows indexed by the elements in $\mathbb{F}^k$. The value of the matrix at the position indexed by the monomials $x_{1,i_i} \cdots x_{k,i_k}$, and the assignment ρ is the value of the monomial $x_{k+1,i_{k+1}} \cdots x_{d,i_d}$ at ρ.*

This definition can be extended to the case when f is not a set-multilinear function. In such a case, the rows of the matrix E_k will be indexed by the monomials of degree d over the first k variables and the columns of the generalized Vandermonde matrix will similarly be indexed by the monomials of degree d.

Observation 3.8. *Let f be a set-multilinear polynomial on the variables*

$X = \dot{\bigcup}_{i=1}^{d} X_i$ *over a field* $\mathbb{F}$*. Then* $\dim(\partial_k(f))$ *is atmost the rank of the matrix* E_k .

This is because each of the rows in the evaluation matrix is the value at each point of $\mathbb{F}^{d-k}$ of the order k partial derivative of f corresponding to that monomial. Thus the number of linearly independent rows in it is atmost the dimension of the space spanned by the partial derivatives.

Now we will see how we can connect the rank of the Hankel matrix of a function to the dimension of the space of partial derivatives. The following lemma which is a generalization of the Taylor series will be useful for it.

Lemma 3.9. *Let* f *be a polynomial in the variables* $x_1, \cdots, x_n$ *over the field* $\mathbb{F}$ *of degree* d*. Consider the assignment* $\rho = \rho_1 \cdot \rho_2$*, where* ρ_1 *is an assignment to the first* k *variables and* ρ_2 *is the assignment to the last* $n-k$ *variables. For a monomial* $M = \prod_{i=1}^{n} x_i^{e_i}$*, where* $\sum_{i=1}^{n} e_i = d$*,* $M(\rho)$ *is the value of the monomial at the assignment* ρ*. We have*

$$f(x_1, \cdots, x_n) = \sum M(\rho_1)\frac{\partial f}{\partial M}(0 \cdot \rho_2)$$

where M *is of the form* $\prod_{i=1}^{k} x_i^{e_i}$*, where* $\sum_{i=1}^{k} e_i = d$ *and* ∂M *refers to* $\partial x_1^{e_1} \cdots \partial x_k^{e_k}$*.*

Proof. Let $f = \sum c_m m$ where m is a monomial of the form $\prod_{i=1}^{n} x_i^{d_i}$. Consider a monomial $M = \prod_{i=1}^{k} x_i^{e_i}$ where $\sum_{i=1}^{k} e_i = d$. If $d_i < e_i$, then $\frac{\partial m}{\partial M} = 0$. But if $d_i > e_i$, then $\frac{\partial m}{\partial M}(0 \cdot \rho) = 0$. Thus only the derivative with respect to the monomial m will survive. This gives the lemma. $\square$

We can bound the rank of the Hankel matrix of a function by the dimension of the space spanned by the partial derivatives. The bound is proved by showing that the Hankel matrix can be expressed as the product of the evaluation matrix and the coefficient matrix. Thus any class of circuits which give small-dimensional space of partial derivatives can be learned using the algorithm of [BBB$^+$00] discussed in the last section.

Theorem 3.10 ([KS06]). *Let* $f(x_1, \cdots, x_n)$ *be a polynomial of degree* d *over a field* F. *Let* H *be the Hankel matrix of* f*, where we consider* $f : \mathbb{F}^n \to \mathbb{F}$*(i.e we consider the alphabet* Σ *in the definition of the Hankel matrix to be the field* $\mathbb{F}$*). Then for* $k \leqslant n$*,*

$$\mathrm{rank}(H_k) \leqslant \dim(\partial_k(f))$$

Proof. Let us look at the Hankel matrix H_k of f. The rows of this matrix are indexed by elements in $\mathbb{F}^k$ ordered lexicographically and the columns are indexed by elements in $\mathbb{F}^{n-k}$. We can show that $H_k = V_{k,d} \cdot E_k$. This follows directly from the lemma we proved earlier and the definition of the matrices $V_{k,d}$ and E_k. The generalized Vandermonde matrix is of full rank and thus

$$\mathrm{rank}(H_k) = \mathrm{rank}(E_k) \leqslant \dim(\partial_k(f))$$

□

From the above theorem we get the following connection between the rank of H and the dimension of the space of partial derivatives

$$\mathrm{rank}(H) \leqslant \sum_{k=0}^{n} \dim(\partial_k f)$$

If we can prove the generalization of the Taylor's theorem that we proved for a general polynomial in the case of set-multilinear polynomials, then we can get a theorem similar to the one above. Let us look at the evaluation matrix E_k to see how the rows of the matrix will be when the polynomial f is set-multilinear over the variables $X = \dot{\bigcup} X_i$. Consider a row that is indexed by a monomial that is not set-multilinear. Let M be such a monomial.

Suppose that the k variables cover the partitions $X_1, \cdots, X_i$ fully and covers part of X_{i+1}. M will contain atleast one variable from each of the parts since otherwise $\frac{\partial f}{\partial M}(0 \cdot \rho_2)$ will evaluate to zero. If it contains more than one variable from a part, then one of the variables will still remain in $\frac{\partial f}{\partial M}$ which will become zero in the assignment $(0 \cdot \rho_2)$. Thus all the rows in E_k that do not correspond to set-multilinear monomials are zero. Thus when we have f as a set-multilinear polynomial we can use the theorem above with slight modifications to show that

Theorem 3.11 ([KS06]). *Let f be a set-multilinear polynomial over the variables $X = \dot{\bigcup}_{i=1}^{d} X_i$. Let H be the Hankel matrix of f where the alphabet Σ is the field $\mathbb{F}$. We have*

$$\text{rank}(H) \leqslant n \cdot \sum_{i=0}^{n} \dim(\partial_i(f))$$

We can now give learning algorithms in the cases when the circuits give a small dimensional space of partial derivatives.

3.3.1 Depth 3 Circuits

As an application of the theorem that we saw just now, let us consider depth 3 set-multilinear circuits. We will assume that the circuits are $\Sigma\Pi\Sigma$. Let the size of the circuit be s. Then there are atmost s multiplication gates in the circuit. From [NW97], we can see that $\dim(\partial_k f) \leqslant s$ and this in combination with previous theorem, we have

Theorem 3.12 ([KS06]). *If C is a set-multilinear depth 3 circuit of size s with n variables over a finite field $\mathbb{F}$, then C is learnable in time polynomial in $n, s, \mathbb{F}$.*

In the case of general depth 3 circuits, the running time of the learning algorithms based on [BBB+00] is worse than in the set-multilinear case.

We have the following bound on the dimension of the space spanned by the partial derivatives

Lemma 3.13. *Let $f(x_1, \cdots, x_n)$ be a polynomial of degree d computed by a circuit of depth 3 of size s. Assume that the number of multiplication gates is m. Then*

$$\dim(\partial_k f) = m \cdot \sum_{i=1}^{k} \binom{d}{i}$$

Proof sketch. Taking k^{th} order partial derivative of f amounts to taking k^{th} partial derivatives of the polynomials at each of the multiplication gates. The polynomial computed at any multiplication gate is the product of affine forms. Thus taking partial derivatives of order k is like choosing k affine forms from the input to that multiplication gate. Since there are m multiplication gates we have the lemma $\square$

With this lemma in hand, and the theorem from this section, we have

Theorem 3.14 ([KS06]). *Let $f : \mathbb{F}^n \to \mathbb{F}$ be computed by a depth 3 circuit of size s and fan-in of the multiplication gates atmost d. Then f is learnable in time polynomial in $n, 2^d, s$.*

3.4 Learning constant depth arithmetic circuits

In the last section, we saw the application of the technique of partial derivatives to get learning algorithms for depth 3 circuits. The model allowed for membership queries from the field. We can show that if we are allowed queries from a matrix algebra, then we have efficient learning algorithms for the class of constant depth circuits. This can be shown by constructing small multiplicity automaton for these functions.

The methods developed in [KS06] cannot be applied for general depth 3 circuits, since we could have small depth 3 circuits that compute polynomials which have a large dimensional space of partial derivatives. For instance the symmetric polynomial Sym_n^d can be computed by small depth 3 circuits but the dimension of the space spanned by the partial derivatives is $\Omega\left((\frac{n}{4d})^d\right)$.

Consider a polynomial $f(x_1, \cdots, x_n)$ computed by a commutative depth D circuit of size s. We can explicitly order the variables from x_1 to x_n and look at the polynomial as a non-commutative polynomial g over $\mathbb{F}\langle x_1, \cdots, x_n\rangle$. For each assignment of values to the variables both f and g evaluate to the same field element. Given such a non-commutative constant depth circuit, we can convert the circuit into a formula of size $O(s^D)$ by opening the circuit up. We have also seen that non-commutative formulas can be converted to ABPs with only a polynomial blow-up in size. Thus for a constant depth circuit, we have a small multiplicity automaton computing the function $g : X^* \to \mathbb{F}$.

Now we will show how we can learn constant depth circuits, represented in the form above when we can query matrices instead of just field elements. Consider a monomial of the form $m = \prod_{k=1}^{d} x_{i_k}$ where the polynomial is of degree d. Corresponding to this monomial, we construct the following automaton A.

1. A has d states labelled from i_1 to i_d.

2. The start state is i_1 and the final state is i_d.

3. There is a transition from a state i_k to i_{k+1} labelled by X_{i_k}.

With this automaton in hand, we can define a $d \times d$ matrix corresponding to A, where $(i, j)^{th}$ entry in the matrix is the label of the transition from i to j. This matrix with each of the variable entries replaced by 1 will be given as the input for the black-box which returns a matrix. By the construction of these matrices the $(i, d)^{th}$ entry will be the coefficient of the monomial m.

The second question that we will have to address here is how to construct a counter-example. In our case, we are given an assignment $a = (a_1, \cdots, a_n)$ such that $f(a) \neq h(a)$ where h is our hypothesis, and from this we want to construct a monomial where the two polynomials differ. Obviously such a monomial must exist, since if all the monomials are the same then the polynomials are the same. To find such a monomial write f and h as follows

$$f(x_1, \cdots, x_n) = x_1 f_1(x_1, \cdots, x_n) + f_2(x_1, \cdots, x_n)$$
$$h(x_1, \cdots, x_n) = x_1 h_1(x_1, \cdots, x_n) + h_2(x_1, \cdots, x_n)$$

Let a be the assignment such that $f(a) \neq h(a)$. We have a two state automaton that recognizes only those monomials that start with x_1. Using this, we can query the black-box to obtain the function f_1 and from it f_2. Similarly from the automaton for h that we have, we can compute both h_1 and h_2. Now if $f_1(a) \neq h_1(a)$, then surely x_1 is the first variable in the monomial which is the counter-example. We can thus repeat this process iteratively to obtain the entire monomial.Now we can apply the learning algorithm of [BBB+00] to construct the multiplicity automaton for the function.

The discussion here has been to show that the method of bounding the size of the automaton by the dimension of the space of partial derivatives need not be the only way to approach the learning theory question for arithmetic circuits.

3.5 Summary

The ideas surveyed in this chapter uses the dimension of the space spanned by the partial derivatives to argue whether a class is efficiently learnable or not. In addition, the learning algorithms discussed have all been improper learning algorithms. This means that even though we were designing a learning algorithms for the class of depth 3 circuits, the algorithm came up with a multiplicity automaton which computed the function. Thus it is worthwhile to ask whether there are "proper" learning algorithms. Note also that [KS06] uses the algorithm from [BBB$^+$00] as a black-box to come up with a class of algorithms. It is also interesting to ask whether there are other measure which can prove useful in designing efficient learning algorithms. We have also seen that if we are allowed to query from a larger algebra, rather that just the field elements, then we can learn the class of constant depth arithmetic circuits.

Chapter 4

Polynomial Identity Testing

4.1 Preliminaries

In this chapter we are interested in the problem of Polynomial Identity Testing(PIT). We are interested in looking at this problem since improvements in the lower bounds for various circuits have led to developing good identity testing algorithms for the same and techniques that have been useful in one problem has found application in the other. The connections between the two are worth exploring, even though we have not managed to formalize this question.

The problem of PIT can be stated thus: *Given a polynomial* $p(x_1, \cdots, x_n)$ *in a succinct manner, say as a circuit, test if* $p \equiv 0$. In this thesis, we will concern ourselves with the case when the polynomial is given as an arithmetic circuit.

The question stated above can be interpreted in two different ways

1. *Given a polynomial* $p(x_1, \cdots, x_n)$ *as a arithmetic circuit, is the formal expression computed by the circuit identically zero.*

2. *Given a polynomial* $p(x_1, \cdots, x_n)$ *as an arithmetic circuit over a field* $\mathbb{F}$, *is* $p(a_1, \cdots, a_n) = 0, \forall (a_1, \cdots, a_n) \in \mathbb{F}^n$.

In the case when $|\mathbb{F}|$ is less than the degree of the polynomial, both these questions are identical.

Efficient deterministic algorithms are not known for this problem. The importance of the problem comes from the fact that connections have been made between finding efficient deterministic algorithms for this problem and proving circuit lower bounds[KI04]. The problem has a very simple randomized algorithm obtained from this lemma

Lemma 4.1 (Schwartz-Zippel Lemma[Sch80, Zip79]). *Let $p(x_1, \cdots, x_n)$ be a polynomial of degree d over a finite field $\mathbb{F}$. Assume that the $d < |\mathbb{F}|$ and that p is not identically zero. Then*

$$\Pr_{x_1,\ldots,x_n \in_R \mathbb{F}}\left[p(x_1, \ldots, x_n) = 0\right] \leqslant \frac{d}{|\mathbb{F}|}$$

The algorithm for identity testing is now simple. Pick $x_1, \cdots, x_n$ uniformly at random from the field $\mathbb{F}$, evaluate the circuit and answer yes if the circuit evaluates to zero. The catch here is that since the circuit is a succinct representation of a polynomial of large degree and coefficients, the values that have to calculated at different stages might be large. This problem can be bypassed by picking a prime p and doing the computations on the circuit modulo p. It can be proved that with high probability this will give the correct answer.

Since identity testing of general circuits are difficult, it is reasonable to ask if there are algorithms for PIT when the circuit is restricted. In this chapter, we survey one such result due to Raz and Shpilka who show that there is a deterministic polynomial time algorithm the circuit is a non-commutative formula.

In the case of depth-3 circuits the best algorithm is the deterministic polynomial time algorithm of [KS07] for $\Sigma\Pi\Sigma$ circuits with bounded top fan-in. For a recent survey on the results and techniques used in identity testing refer to [Sax09].

In this chapter, we will look at non black-box identity testing. Here we are allowed to use the underlying structure of the circuit which we are testing. Contrast to this, in the case of black-box identity testing the algorithm is allowed to query the circuit like a black-box.

Since in this thesis, we are more interested in the partial derivative method, we will go into the discussion of the algorithm of [RS05].

4.2 PIT in Non-commutative models

From the identity testing algorithm for non-commutative formulas we will also get an identity testing algorithm for depth 3 set-multilinear circuits as a corollary.

We will prove that

Theorem 4.2. *Given a non-commutative formula* F *of size* s_F*, we can decide in time polynomial in* s_F *if the polynomial computed by* F *as a formal expression is zero or not.*

As a corollary we also get that

Corollary 4.3. *Given a depth 3 set-multilinear circuit* C *over the variables* $X = \dot{\bigcup}_{i=1}^{d} X_i$ *of size* s_C*, we can test deterministically in time polynomial in* s_C *whether the polynomial computed by* C *is identically zero or not.*

Recall that a formula is a class of arithmetic circuit where each gate has fan-out exactly one except the output gate. A non-commutative formula is one where the inputs to the various gates are ordered and the formula computes a polynomial in the free algebra $\mathbb{F}\langle x_1, \dots, x_n\rangle$. [Nis91] showed that the vector space spanned by the partial derivatives of functions computed by non-commutative formulas has small dimension. [RS05] give a method for recursively checking if the all the partial derivatives are zero.

The approach of [Nis91] was to show that non-commutative formulas can be converted to arithmetic branching programs with only a polynomial blow-up in size. [RS05] work with these arithmetic branching programs to check if it is identically zero.

The basic idea is to reduce the identity testing problem on the given ABP to identity testing on an ABP which has a lesser number of levels with the same size. Recall Lemma 2.17 where we converted a non-commutative formula into a homogeneous ABP. In the lemma we showed that the non-homogeneous ABP can be converted to atmost $d+1$ homogeneous ABPs each of size $O(s^2)$. It can also be seen from the proof that the algorithm runs in time polynomial in s. Thus without loss of generality, we will assume that the ABP that we identity test is homogeneous.

For an homogeneous ABP A with source s and sink t, with nodes $v_1, \cdots, v_m$ in the second layer we can write

$$A_{s,t} = \sum_{i=1}^{m} A_{s,v_i} \cdot A_{v_i,t}$$

We know that A_{s,v_i} has monomials of the form $\alpha \cdot x_i^2$ or $\alpha \cdot x_i x_j$. The idea of [RS05] is to replace these with a different set of linear forms thus reducing the number of levels to get a new ABP $\hat{A}$ such that $A_{s,t} \equiv 0 \Leftrightarrow \hat{A}_{s,t} \equiv 0$.

A straightforward way in which we can replace the quadratic forms by linear forms is to replace each pair of variables $x_i x_j$ by a new variable y_{ij}. This way we introduce atmost n^2 new variables when we reduce the number of layers by 1. Since the ABP could have n^c many layers, this could increase the number of variables to an exponential value. Hence the straightforward way is not good enough if we want a polynomial time algorithm for identity testing.

[RS05] provide a cleverer way to label the edges between the two levels by introducing just $O(S+n)$ many variables where S is the size of the ABP. As was noted earlier, the polynomials computed in the first two layers of the ABP are quadratic forms. Thus we can write

$$A_{s,v_i} = \sum_{j=1}^{n} \alpha_{ij} x_j^2 + \sum_{1 \leqslant j < k \leqslant n} \beta_{ijk} x_j x_k + \sum_{1 \leqslant j < k \leqslant n} \gamma_{ijk} x_k x_j$$

Since we know that $A_{s,t} = \sum_{i=1}^{m} A_{s,v_i} \cdot A_{v_i,t}$, we have

$$A_{s,t} = \sum_{j=1}^{n} x_j^2 \sum_{i=1}^{m} \alpha_{ij} A_{v_i,t} + \sum_{1 \leqslant j < k \leqslant n} x_j x_k \sum_{i=1}^{m} \beta_{ijk} A_{v_i,t} + \sum_{1 \leqslant j < k \leqslant n} x_k x_j \sum_{i=1}^{m} \gamma_{ijk} A_{v_i,t}$$

Since we are working in the non-commutative domain, a polynomials which has x_i^2 in the beginning cannot cancel one which has $x_i x_j, i < j$ in the beginning. Thus we have the easy observation that

Observation 4.4. $A_{s,t} \equiv 0$ *iff*

$$\begin{aligned} \forall 1 \leqslant j \leqslant n \sum_{i=1}^{m} \alpha_{ij} A_{v_i,t} &\equiv 0 \\ \forall 1 \leqslant j < k \leqslant n \sum_{i=1}^{m} \beta_{ijk} A_{v_i,t} &\equiv 0 \\ \sum_{i=1}^{m} \gamma_{ijk} A_{v_i,t} &\equiv 0 \end{aligned}$$

Recall Lemma 2.18 in which it was shown that $\dim(\partial(p))$ is atmost S where p is the polynomial computed by an ABP of size S. Now if you look at partial derivatives of order 2, the proof of the lemma gives you the fact that $\dim(\partial_2(p))$ is atmost m where m is the number of nodes in the second layer.

With this is mind, define the vectors

$$\begin{aligned}\forall 1 \leqslant j \leqslant n\ \alpha_j &= (\alpha_{1,j}, \cdots, \alpha_{m,j})\\ \forall 1 \leqslant j < k \leqslant n \beta_{j,k} &= (\beta_{1,j,k}, \cdots, \beta_{m,j,k})\\ \forall 1 \leqslant j < k \leqslant n \gamma_{k,j} &= (\gamma_{1,k,j}, \cdots, \gamma_{m,k,j})\\ \overrightarrow{a} &= (A_{v_1,t}, \cdots, A_{v_m,t})\end{aligned}$$

From the discussion in the last paragraph, we know that not all of these are linearly independent. Note that $A_{s,t} \equiv 0$ iff $\alpha_i \cdot \overrightarrow{a} = 0$ for all $i \in [n]$ and $\beta_{jk} \cdot \overrightarrow{a} = \gamma_{kj} \cdot \overrightarrow{a} = 0$ for $j < k \in [n]$.

Let d be the dimension of the space spanned by $\alpha_i, \beta_{jk}, \gamma_{kj}, i \in [n], j < k \in [n]$. Since there are m nodes in the second layer of the ABP A, $d \leqslant m$. Let $u_1, \cdots, u_d$ denote the basis vectors where

$$u_i = (u_{i,1}, \cdots, u_{i,m})$$

Define linear forms l_i for each $i \in [m]$ as

$$l_i(y_1, \cdots, y_d) = \sum_{j=1}^{d} u_{ji} \cdot y_j$$

Now we replace all the layer 1 nodes of the ABP A and connect the source vertex of A, s to all the nodes in layer 2 where the label of the edge connecting s with node v_i is l_i. This new ABP will be $\hat{A}$.

Claim. $A \equiv 0 \Leftrightarrow \hat{A} \equiv 0$

Proof. The polynomial computed by the ABP $\hat{A}$ is

$$\hat{A}_{s,t} = \sum_{i=1}^{m} l_i \cdot A_{v_i,t}$$

From the definition of the linear forms l_i, we get

$$\begin{aligned}\hat{A}_{s,t} &= \sum_{i=1}^{m}\Big(\sum_{j=1}^{d} u_{ji}y_j\Big)\cdot A_{v_i,t}\\ &= \sum_{j=1}^{d} y_j\Big(\sum_{i=1}^{m} u_{ji}A_{v_i,t}\Big)\\ &= \sum_{i=1}^{d} y_i(u_i\cdot\overrightarrow{a})\end{aligned}$$

Thus $\hat{A}_{s,t} \equiv 0$ if and only if $u_i \cdot \overrightarrow{a} = 0, \forall i \in [m]$. This is true iff the space spanned by u_is give a product 0 with $\overrightarrow{a}$. This gives the claim. □

Thus we have the following algorithm for identity checking a non-commutative arithmetic formula F.

1. Convert the arithmetic formula F which computes a degree d polynomial $p \in \mathbb{F} \in \langle x_1, \cdots, x_n\rangle$ to $d+1$ ABPs $A_0, \cdots, A_d$ where A_i computes the $i-1^{th}$ homogeneous part of the polynomial p.

2. For each A_i, reduce the number of layers of A_i by one by introducing new variables and connecting layers 0 and 2 with a new set of linear forms. Continue doing this till we get a 2 layered ABP on which we can identity test

The time taken to convert the formula into $d+1$ ABPs is polynomial in the size s_F of the formula F. Each ABP is of size $O(s_F^2)$. To compute the all the values $\alpha_i, \beta_{jk}, \gamma_{kj}$, we require time polynomial in m, m', n where m is the number of nodes in the second layer and m' is the number of nodes in the first layer. Thus we have n^2+n vectors and we need to find the set of linearly independent vectors among this that will span the vector space spanned by the n^2+n vectors. This can be done in polynomial time. This would give the linear forms l_is and we can convert the ABP A into a new one $\hat{A}$ which has one level less. We need to iterate this procedure atmost d times to get to a 2 layered ABP A' which computes degree 2 polynomials such that $A \equiv 0 \Leftrightarrow A' \equiv 0$. Thus the entire procedure runs in time polynomial in s_F. This proves Theorem 4.2.

4.2.1 Depth 3 Set-Multilinear Circuits

The technique described above for identity testing non-commutative formulas lends an algorithm for identity testing depth 3 set-multilinear circuit. Recall that a depth 3 $\Sigma\Pi\Sigma$ set-multilinear circuit, computes a set-multilinear function over some fixed partition of variables at every gate of the circuit. Firstly, note that any depth 3 $\Sigma\Pi\Sigma$ circuit C can be thought of as an ABP A as follows.Assume that C computes a polynomial

$$p(x_1, \cdots, x_n) = \sum_{i=1}^{m_1} \prod_{j=1}^{m_2} L_{ij}(x_1, \cdots, x_n)$$

where L_{ij}s are linear functions.

1. The number of layers will be the maximum fan-in of the $\times$ gates. Order the linear forms input to a $\times$ gate an put each linear form in a different layer.

2. Introduce two nodes s and t as the source and sink respectively, add a path from s to t for each $\times$ gate where the length of the path is the fan-in of the gate and the edge labels are the linear forms.

The technique used in the last section cannot be extended in the case of these general circuits since we cannot split the polynomial into 3 parts as we did there. Set-multilinear circuits on the other hand have nice properties which enable us to use the same algorithm as in the case of non-commutative formulas to obtain a deterministic identity testing algorithm.

We will further assume that the circuit is a $\Sigma\Pi\Sigma$ circuit. The case of $\Pi\Sigma\Pi$ circuit is not interesting since any set-multilinear circuit on variables $X = \dot{\bigcup}_{i=1}^{d} X_i$ will have fan-in atmost d for the top Π gate. Each Σ gate will compute a polynomial from one part.

In the case of $\Sigma\Pi\Sigma$ circuit, since in any set-multilinear circuit each gate computes a set-multilinear polynomial, it is necessary that the linear function computed by a $+$ gate at the bottom level depends on just one part of the variable partition. Thus we can think of the commutative set-multilinear circuit as a non-commutative circuit. Every monomial in the polynomial can be thought of as ordered from $X_1, \cdots, X_d$ since only one variable from each part will occur in a monomial. This gives us the corollary 4.3

4.3 Summary

[RS05] show that the same techniques as was used in the design of a deterministic algorithm can be used to design algorithms for PIT in the case of pure circuits. There are a special class of set-multilinear circuits where the set of variables that are used in the computation of polynomials at two different gates are either same or mutually disjoint. The most naive circuit for iterated matrix multiplication is of this form. [NW97] have also shown depth lower bounds for such circuits for computing the iterated matrix product.

The question of connection between circuit lower bounds and PIT is interesting. Non-trivial lower bounds in the size or depth has been usually followed by algorithms for identity testing in the particular class of circuits.

Chapter 5

Conclusion

The aim of this thesis had been to survey a technique that has been useful in proving lower bounds for arithmetic circuits, designing efficient learning algorithms for certain classes of arithmetic circuits and for identity testing arithmetic circuits.

First, we must state that the list of results that we have surveyed which use these techniques is not exhaustive. [SW01] use these techniques to prove lower bounds for general circuits. It has also been used by [Raz04] and [Raz09] to prove results about multilinear circuits.

A question that is interesting and worth further exploration is the connection between the problems of proving lower bounds, designing efficient learning algorithms and constructing good identity testing algorithms. It is easy to see that the method of [BBB$^+$00] gives an efficient identity testing algorithm, if there are no equivalence queries since we can use [RS05] on the multiplicity automaton. Similarly, proving circuit lower bounds is an easier problem than designing efficient algorithms for identity testing. Even though we have seen similar techniques used in these three settings, we do not know of a way to quantitatively compare the hardness of these problems and feel that this is an intersting avenue for further research.

Bibliography

[Ang87] D. Angluin. *Queries and concept learning.* Machine Learning, 2(4):319–342, 1987.

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

[BR88] Jean Berstel and Christophe Reutenauer. *Rational Series and Their Languages.* Number 12 in EATCS Monographs on Theoretical Computer Science. Springer-Verlag, 1988.

[BS83] W. Baur and V. Strassen. *The complexity of partial derivatives.* Theoretical Computer Science, 22(3):317–330, February 1983. doi:10.1016/0304-3975(83)90110-X.

[CP71] J. W. Carlyle and A. Paz. *Realizations by stochastic finite automata.* Journal of Computer and System Sciences, 5(1):26–40, February 1971.

[CS07] Steve Chien and Alistair Sinclair. *Algebras with polynomial identities and computing the determinant.* SIAM Journal of Computing, 37(1):252–266, 2007. doi:10.1137/S0097539705447359.

[Fli74] Michel Fliess. *Matrices de Hankel.* Journal des Mathématiques Pures et Appliquées, 53:197–224, 1974.

[FSS84] Merrick L. Furst, James B. Saxe, and Michael Sipser. *Parity, circuits, and the polynomial-time hierarchy.* Mathematical Systems Theory, 17(1):13–27, 1984.

[GK98] Dima Grigoriev and Marek Karpinski. *An exponential lower bound for depth 3 arithmetic circuits.* In *STOC*, pages 577–582. 1998. doi:10.1145/276698.276872.

[Got66] D. H. Gottlieb. *A certain class of incidence matrices.* Proceedings of the American Mathematical Society, 17(6):1233–1237, 1966.

[GR98] Dima Grigoriev and Alexander A. Razborov. *Exponential complexity lower bounds for depth 3 arithmetic circuits in algebras of functions over finite fields.* In *FOCS*, pages 269–278. 1998. doi:10.1109/SFCS.1998.743456.

[Hås86] Johan Håstad. *Almost optimal lower bounds for small depth circuits.* In *STOC*, pages 6–20. ACM, 1986.

[HR90] T. Hagerup and C. Rub. *A guided tour of Chernov bounds.* Inform. Proc. Lett., 33:305–308, 1990.

[KI04] Valentine Kabanets and Russell Impagliazzo. *Derandomizing polynomial identity tests means proving circuit lower bounds.* Computational Complexity, 13(1-2):1–46, 2004. doi:10.1007/s00037-004-0182-6.

[KS06] A. R. Klivans and A. Shpilka. *Learning restricted models of arithmetic circuits*. Theory of Computing, 2(1):185–206, 2006. doi:10.4086/toc.2006.v002a010.

[KS07] Neeraj Kayal and Nitin Saxena. *Polynomial identity testing for depth 3 circuits*. Computational Complexity, 16(2):115–138, 2007. doi:10.1007/s00037-007-0226-9.

[Nis91] N. Nisan. *Lower bounds for non-commutative computation (extended abstract)*. In *STOC*, pages 410–418. ACM, 1991. doi:10.1145/103418.103462.

[NW97] N. Nisan and A. Wigderson. *Lower bounds on arithmetic circuits via partial derivatives*. CMPCMPL: Computational Complexity, 6, 1997. doi:10.1007/BF01294256.

[Raz87] Razborov. *Lower bounds on the size of bounded depth circuits over a complete basis with logical addition*. MATHNASUSSR: Mathematical Notes of the Academy of Sciences of the USSR, 41, 1987.

[Raz04] R. Raz. *Multilinear-NC_1 != multilinear-NC_2*. Electronic Colloquium on Computational Complexity (ECCC), (042), 2004.

[Raz05] Ran Raz. *Lower bounds for algebraic circuits*, 2005. Barbados Workshop on Computational Complexity.

[Raz09] R. Raz. *Multi-linear formulas for permanent and determinant are of super-polynomial size*. J. ACM, 56(2), 2009. doi:10.1145/1502793.1502797.

[RR97] Alexander A. Razborov and Steven Rudich. *Natural proofs*. J. Comput. Syst. Sci., 55(1):24–35, 1997.

[RS05] R. Raz and A. Shpilka. *Deterministic polynomial identity testing in non-commutative models*. CMPCMPL: Computational Complexity, 14, 2005.

[Sax09] N. Saxena. *Progress on Polynomial Identity Testing*. Bull. EATCS, 99:49–79, 2009.

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

[Shp02] Amir Shpilka. *Affine projections of symmetric polynomials*. J. Comput. Syst. Sci, 65(4):639–659, 2002. doi:10.1016/S0022-0000(02)00021-1.

[Smo87] Roman Smolensky. *Algebraic methods in the theory of lower bounds for Boolean circuit complexity*. In *Proceedings of the Nineteenth Annual ACM STOC*, pages 77–82. New York City, 25–27 May 1987.

[Str73] V. Strassen. *Die berechnungskomplexetat von elementarysymmetrischen funktionen und von interpolations koeffizienten*. Numer. Math., 20:238–251, 1973.

[SW01] A. Shpilka and A. Wigderson. *Depth-3 arithmetic circuits over fields of characteristic zero*. CMPCMPL: Computational Complexity, 10, 2001. doi:10.1007/PL00001609.

[Val84] L. G. Valiant. *A theory of the learnable*. Communications of the ACM, 27:1134–1142, 1984.

[Val92] Leslie G. Valiant. *Why is boolean complexity theory difficult?* In Mike S. Paterson, ed., *Boolean function com-*

plexity, volume 169 of *London Mathematical Society Lecture Note Series*, pages 84–94. Cambridge University Press, Cambridge, 1992.

[Zip79] RICHARD ZIPPEL. *Probabilistic algorithms for sparse polynomials.* In EDWARD W. NG, ed., *EUROSAM*, volume 72 of *Lecture Notes in Computer Science*, pages 216–226. Springer, 1979.