Lower Bounds for Non-Commutative Computation
Extended Abstract

Noam Nisan *

Abstract

We consider algebraic computations which are not allowed to rely on the commutativity of multiplication. We obtain various lower bounds for algebraic formula size in this model: (1) Computing the determinant is as hard as computing the permanent and tight exponential upper and lower bounds are given. (2) Computation cannot be parallelized, as opposed to in the commutative case – this solves in the negative an open problem of Miller et al [8]. (3) The question of the power of negation in this model is shown to be closely related to a well known open problem relating communication complexity and rank.

We then take modest steps towards extending our results to general, commutative algebraic computation, and prove exponential lower bounds for monotone algebraic circuit size, as well as for the size of certain types of constant depth algebraic circuits.

1 Introduction

1.1 Overview

Algebraic complexity theory investigates the complexity of computing algebraic functions using algebraic operations, usually addition, subtraction, multiplication, and division. The algebraic computation is usually described by a *straight line program* or, equivalently as we will, by an *algebraic circuit.* Much research has been done designing efficient algebraic computations for various functions and that research has achieved remarkable success and has produced many clever algorithms. On the other hand, the equally large effort put into trying to prove *lower bounds* to algebraic complexity has achieved very little. To date, the best lower bound known for the size of algebraic circuits computing any (explicit, polynomial degree) function is $\Omega(n \log n)$. The only lower bound known for algebraic circuit depth is the trivial $\Omega(\log n)$. (See, e.g. [3] for a survey)

This situation is very similar to the situation in boolean complexity theory where essentially no nontrivial lower bounds are known. In boolean complexity theory the line of attack taken by the community is to prove lower bounds for *restricted* circuits. Specifically, exponential lower bounds are known for constant depth circuits and for monotone circuits (see [2] for a survey). Similarly, restricted algebraic circuits may be considered. Strassen [11] shows that restricting the circuits not to use division gates does not change (by more than a polynomial factor) the complexity of computing any multivariate polynomial. Monotone algebraic circuits (i.e. circuits using only addition and multiplication and only positive constants) have been investigated about a decade ago, and exponential lower bounds are known for several functions [12, 13, 14, 6, 9]. For constant depth algebraic circuits no lower bounds are known.

In this paper we consider another restriction, one of an algebraic flavor: we require the circuits not to use the fact that multiplication is commutative. Formally, we work in

*Department of Computer Science, Hebrew University, Jerusalem. Supported by BSF 89-00126. Part of this work was done while visiting MIT, supported by NSF 8912586-CCR.

the non-commutative ring $R\{x_1 \dots x_n\}$ where the indeterminates x_i do not commute with each other instead of, as usual, in the commutative ring $R[x_1 \dots x_n]$. We call such circuits *non-commutative circuits*. This restriction has already been studied. For the case of bilinear forms, Winograd [15] shows that restricting the circuits not to rely on commutativity at most doubles the complexity. Indeed, all of the work on matrix multiplication (which is a bilinear form) is implicitly done using only non-commutative circuits. Hyafil [4] considers the computation of the determinant, and proves that non-commutative circuits that compute the determinant require exponential size. Unfortunately, Hyafil's proof is incorrect[1]. Although we conjecture that the lower bound is still correct, we can only prove a weaker statement, namely that the determinant requires exponential *formula* size with non-commutative circuits.

In this paper we study several complexity questions regarding non-commutative circuits. Most of our work we do using a complexity measure that can be viewed as the algebraic analogue of branching program size. We first show that some standard simulations in the commutative case extend to the non-commutative case, and thus lower bounds for this complexity measure also give lower bounds for formula size. Conversely, we show, that the gap in the other direction is also not too big. We then give a simple characterization for the complexity of a function in this measure in terms of the combinatorial structure of the monomials of the function. From this we obtain several results.

1.2 Results regarding non-commutative computation

Permanent and Determanent

We prove that the permanent and the determinant functions have exactly the same complexity in this measure. We then translate our bounds to the usual measures of circuit size and depth and formula size. The upper bound we obtain is mostly of interest for the permanent function.

Theorem: The permanent function can be computed by non-commutative algebraic circuits of size $O(n2^n)$. The same is true for the determinant function.

This matches the best algorithm known for the computation of the permanent even in the commutative case. The lower bound we obtain is of interest mostly for the determinant function.

Theorem: The determinant function requires $2^{\Omega(n)}$ non-commutative formula size. The same bound holds for the permanent function.

This should be contrasted with the fact that in the commutative case the determinant can be computed by formulas of size $n^{O(\log n)}$. One's first impression of this contrast is to assume that the efficient commutative formula for determinant computes each of the monomials $\prod_i x_{i,\sigma(i)}$ of the determinant in some different multiplicative order. The situation is actually more complicated. The lower bound mentioned applies to any order of multiplication in each monomial. The point is that the efficient commutative formulas for determinant must rely on *cancellations* between un-needed monomials which differ in the order of multiplication. They must rely on identities such as $xy - yx = 0$, as for example in a computation of $x^2 - y^2$ by $(x + y) * (x - y)$.

Parallelization of code

Hyafil [5] proves that any polynomial size algebraic circuit that computes a multivariate polynomial of polynomial degree can be transformed into an equivalent algebraic circuit of depth $O(\log^2 n)$. Valiant et al [10] prove that this transformation can be done while keeping the size polynomial as well. Miller et al [8] improve the construction and extend it to computation over any commutative semi-ring. They ask whether the same can be done in non-commutative semi-rings or rings. We answer this question in the negative.

Theorem: There exists (an explicitly given) function of degree n which can be computed by a linear size non-commutative circuit but any non-commutative circuit for it requires depth $\Omega(n)$.

Monotone vs. non-monotone computation

[1] The error is in the proof of theorem 2 in the analysis of case 2(a): because of the non-commutativity, the case of $deg(f') \geq d/3$ is different from the case $deg(f) \geq d/3$, and is not dealt with. The statement of theorem 2 itself is false, and the function exhibited here in section 4.2 serves as a counter example.

Valiant [9] exhibits a monotone function (i.e. that all of its monomials have positive coefficients) that can be computed by polynomial size algebraic circuits but any *monotone* algebraic circuit computing it requires exponential size. We ask whether the same difference between monotone and non-monotone also exists for non-commutative computation. surprisingly, it turns out that this question is very closely related to the following question in communication complexity.

Consider a boolean function $b(x, y)$. Denote by $C(b)$ the communication complexity of b and by $rank(b)$ the rank of the real matrix associated with it (i.e. having a row for each possible value of x and a column for each value of y). It is known that for any function b, $C(b) \geq \log_2 r(b)$. Lovasz and Saks (see [7]) ask whether $\log r$ and C are polynomially related, i.e. whether $C(b) = (\log r(b))^{O(1)}$.

We show that this problem is intimately related to questions regarding the power of negation in non-commutative computation. In fact, we give a necessary and sufficient condition for the answer to be "Yes" in terms of certain measures of complexity for monotone vs. non-monotone non-commutative computation.

1.3 Commutative computation

Clearly the main motivation for studying non-commutative computation is to gain insight into the nature of general, commutative computation. We have in fact been able to obtain some weak results in this direction, using our lower bounds for non-commutative computation in order to obtain lower bounds for commutative computation which is restricted in other ways:

Monotone circuits

As mentioned above, the lower bounds we get for the complexity of the permanent hold for any order of multiplication in each monomial. The only way commutative circuits may compute the function more efficiently is by using cancellations of monomials of different order of multiplication. In any monotone circuit there can not be any cancellations at all. It thus follows that monotone circuits can not compute the permanent function in small size, even allowing commutativity. This allows us to duplicate some of the known lower bounds for monotone circuits and obtain a new simple proof for:

Theorem: [13, 14] The monotone formula size of the permanent is $2^{\Omega(n)}$. The monotone circuit size of the permanent is $2^{\Omega(\sqrt{n})}$.

Depth three circuits

Although for boolean circuits lower bounds for constant depth circuits are known, no such bounds are known for algebraic circuits. The simplest types of circuit for which it is not trivial to prove lower bounds is the following type of circuit: it has an addition gate of unbounded fanin at the top; the middle layer has multiplication gates of unbounded fanin; and each of these gates is fed some linear function of the inputs (i.e. another layer of addition gates). So far there is no known lower bound for the size of such depth three circuits. Ben-Or (as well as, undoubtly, others) has shown that they are rather powerful and can in particular compute in polynomial size any symmetric function. This is in direct contrast to the boolean case where the lower bounds known apply also to simple symmetric functions such as majority.

Ben-Or's construction uses two "un-tidy" features: (1) The circuit is not homogeneous (2) Computing a function of degree $d << n$ requires intermediate results of degrees as high as n. In the case of bounded fanin circuits it is always possible to "correct" these problems: it only takes a polynomial penalty in size to convert non-homogeneous circuits to homogeneous ones, and to ones that never use intermediate results of degree more than d ([9, 11]). We prove exponential lower bounds for the size of depth three circuits that are further constrained in one of these ways, showing that (1) Ben-Or's construction cannot be improved in any of these senses and (2) Converting non-homogeneous depth three circuits to homogeneous ones or to ones that do not use intermediate results of high degree requires an exponential blowup in size.

Theorem: Any homogeneous depth three circuit that computes the d'th elementary symmetric function requires size $(n/d^3)^{\Omega(d)}$. Any depth three circuit that computes the d'th elementary symmetric function while using intermediate results of degree at most D requires size

$(n/(D^2 d))^{\Omega(d)}$.

2 Circuits, formulas, and algebraic branching programs

Formally we will be working in the ring $R\{x_1 \ldots x_n\}$, i.e. the ring extension of the reals by the non-commuting indeterminates $x_1 \ldots x_n$. In this ring addition is commutative but multiplication is not. Algebraic circuits and formulas are defined as usual, but the inputs to each multiplication gate are labeled by "left" and "right", specifying the order of multiplication. The operations allowed are addition, subtraction and multiplication, and each has unit cost. Any real constants may be used. It will be easier for us to use a related computation model, which is the analogue of branching programs.

Definition 1 *An* Algebraic branching program (ABP) *is a directed acyclic graph with one source and one sink. The vertices of the graph are partitioned into "levels" numbered from* 0 *to* d*, where edges may only go from level* i *to level* $i+1$. d *is called the degree of the ABP. The source is the only vertex at level* 0 *and the sink is the only vertex at level* d. *Each edge is labeled with a homogeneous linear function of* $x_1 \ldots x_n$ *(i.e. a function of the form* $\sum_i c_i x_i$*). The size of an ABP is the number of vertices.*

An ABP computes a function in the obvious way: the sum over all paths form the source to the sink, of the product of the linear functions by which the edges of the path are labeled. It is clear that an ABP of degree d computes a homogeneous (multivariate) polynomial of degree d.

Definition 2 *The formula complexity of a function* f *is denoted by* $F(f)$*, the circuit complexity by* $C(f)$*, the circuit depth complexity by* $D(f)$*, the ABP complexity by* $B(f)$.

The following lemma relates these complexity measures. The proofs are similar to the proofs of similar facts in the commutative case, and show that the known simulations do not rely on commutativity. They are stated only for homogeneous functions as ABPs can only compute homogeneous functions, but can be easily extended to any function if, e.g. we allow each homogeneous component to be computed by a separate ABP.

Lemma 1 *1. For any homogeneous function* f *of degree* d*:* $B(f) \leq d(F(f)+1)$.

2. For any homogeneous function f*:* $C(f) \leq O(nB(f)^2)$.

3. For any homogeneous function f *of degree* d*:* $D(f) \leq O(\log B(f) \log d)$.

4. For any function f*:* $F(f) \leq 2^{D(f)}$.

Proof:

Proof of 1: (Sketch) The simulation is done in two stages. In the first stage we transform the formula to a non-homogeneous ABP, i.e. an ABP where the nodes are not required to be partitioned into levels, and where each edge may be labeled by a constant or variable. This is done by recursively converting the formula $f_1 + f_2$ to a parallel wiring of non-homogeneous ABPs for f_1 and for f_2, and the formula $f_1 + f_2$ to a sequential wiring. This stage builds a non-homogeneous ABP of size at most $F(f)+1$.

In the second stage we convert this non-homogeneous ABP to a simple ABP. This is done by partitioning the function computed at each vertex to its homogeneous components. This stage may increase the size by at most a factor of d.

Proof of 2: (Sketch) Each vertex v of the ABP is converted to an addition gate of fanin at most $B(f)$. Each edge (u,v) feeding into v is converted to a multiplication gate feeding to that addition gate. The multiplication gate multiplies the function computed at u by the linear form that labels the edge (u,v). Computing the linear form takes $O(n)$ gates, and each addition gate of unbounded fanin converts to $B(f)$ addition gates of fanin 2.

Proof of 3: (Sketch) To simulate the ABP we need to multiply d matrices each of size at most $B(f)$ by $B(f)$. Multiplying two matrices takes depth $O(\log B(f))$, and we multiply all d of them in a binary tree fashion.

Proof of 4: Trivial.

$\square$

3 A characterization of ABP complexity

Let f be a homogeneous function of degree d on n variables. For each $0 \leq k \leq d$ we define a real matrix $M_k(f)$ of dimensions n^k by n^{d-k} as follows: there is a row for each sequence of k variables (called k-term), and a row for each $d-k$-term (sequence of $d-k$ variables, out of the possible n, allowing repetitions). The entry at $< x_{i_1} \dots x_{i_k} >, < x_{j_1} \dots x_{j_{d-k}} >$ is defined to be the real coefficient of the monomial $x_{i_1} \dots x_{i_k} x_{j_1} \dots x_{j_{d-k}}$ in f.

Theorem 1 *For any homogeneous function f of degree d,*

$$B(f) = \sum_{k=0}^{d} rank(M_k(f))$$

Proof: (sketch) Fix an ABP that computes f. For any $0 \leq k \leq d$ let $v_1 \dots v_t$ be the vertices of the ABP in the k'th level. We define two matrices L_k and R_k as follows: L_k will have a row for each k-term and a column for each $0 \leq i \leq t$ and R_k will have a row for each $0 \leq i \leq t$ and a column for each $d-k$-term. For a k-term τ, and i, $L_k[\tau, i]$ is defined to be the coefficient of τ in function computed by the restricted ABP where v_i is the sink. For a $d-k$-term σ, and i, $R_k[i, \sigma]$ is defined to be the coefficient of σ in the function computed by the restricted ABP where v_i is the source. Using this notation, it is easy to verify that $M_k(f) = L_k R_k$.

The lower bound follows since $rank(M_k(f)) \leq rank(L_K) \leq t$, and thus for each k, the number of vertices on the k'th level must be at least $rank(M_k(f))$.

In order to prove the upper bound we will show that if for some level k the number of vertices of the ABP is more than $rank(M_k(f))$ then we can build a smaller ABP that computes f. Again, let t be the number of vertices on level k. If $rank(L_k) < t$ then there exists a column i in L_k that is a linear combination of the others. This implies that the function computed by the restricted ABP where v_i is the sink is a linear combination of the functions computed by the restricted ABPs with the other vertices in this level as sinks. It is not difficult to see that at this point vertex v_i can be eliminated from the ABP, and each edge connecting it to a vertex u in the $k+1$'st level should be replaced by edges connecting u to the appropriate linear combination of other vertices in the k'th level. Similarly, if $rank(R_k) < t$ then we can reduce the size of the ABP. Finally, it is a matter of simple linear algebra to show that if L_k and R_k have a full rank of t then also $M_k(f) = L_k R_k$ has rank t. $\square$

4 Corollaries

4.1 Permanent and determinant

In the non-commutative case we must define the order of multiplication in each monomial of the permanent and determinant.

Definition 3

$$perm(x_{1,1} \dots x_{n,n}) = \sum_{\sigma \in S_n} \prod_i x_{i,\sigma(i)}$$

$$det(x_{1,1} \dots x_{n,n}) = \sum_{\sigma \in S_n} sign(\sigma) \prod_i x_{i,\sigma(i)}$$

Two functions f, g are weakly equivalent *if for each monomial of f with non-zero coefficient there exists a monomial of g composed of the same variables (but perhaps in different multiplicative order) with non-zero coefficient, and vice-versa.*

Lemma 2 *For $0 \leq k \leq n$,*

$$rank(M_k(perm)) = rank(M_k(det)) = \binom{n}{k}$$

Proof: (sketch) After eliminating duplicate rows and columns in $M_k(perm)$ we are left with a matrix having a row for each subset of size k of

$\{1 \ldots n\}$ and a column for each subset of size $n-k$. The value of an entry in the matrix is 1 if the sets describing this entry are disjoint, and 0 otherwise. Thus the matrix is diagonal and has full rank. For the determinant, the non-zero entries can have values $+1$ and -1, inspection reveals that we get each row and each column twice, one of them negated. $\square$

Lemma 3 *For any function f which is weakly equivalent to the permanent (or equivalently to the determinant), and for every $0 \leq k \leq n$,*

$$rank(M_k(f)) \geq \binom{n}{k}$$

Proof: (Sketch) Each row of $M_k(f)$ defines two sets $S_1, S_2 \subset \{1 \ldots n\}$ of size k, S_1 being the set of rows (of the input matrix) that is used in the k-term defining the row, and S_2 the set of columns. Similarly each column defines two sets T_1 and T_2 of size $n-k$. It is clear that an entry of the matrix can be non-zero only if S_1 is the complement of T_1 and S_2 the complement of T_2. Thus if we look at the matrix as composed of blocks (defined according to S_1, S_2, T_1, T_2) we get a matrix that is diagonal "in blocks". The rank of the matrix is thus at least the number of non-all-zero blocks. Let us compute the maximum number of monomials of f that can be obtained from one block. All monomials deriving from a block given by S_1, S_2, T_1, T_2 must be composed of variables defining a permutation mapping S_1 to S_2, and T_1 to T_2. Any such permutation is possible, giving an upper bound of $k!(n-k)!$. We conclude the proof by observing that the total number of monomial that f must have is at least $n!$. $\square$

We can now prove the following theorems:

Theorem 2 *For any function f which is weakly equivalent to permanent or determinant, $F(f) \geq 2^{\Omega(n)}$.*

Proof: Using Theorem 1 sum the lower bounds given by lemma 3 for all values of $0 \leq k \leq n$. Finally, apply lemma 1.1. $\square$

Theorem 3

$$C(perm) \leq O(n2^n)$$
$$C(det) \leq O(n2^n)$$

Proof: (sketch) Direct usage of lemma 1.2 with theorem 1 and lemma 2 gives a slightly weaker bound of $O(n4^n)$. A more delicate study of the ABP suggested by lemma 2, shows that (1) in this case the fanin of each vertex on level k is exactly k, and (2) each edge is labeled by single variable. This suffices for fine-tuning the proof of lemma 1.2 to get the improved bound. $\square$

4.2 Depth vs. size

Let $w \in \{0,1\}^*$. Denote by w^R the reverse of w. Denote by x^w the monomial $\prod_{i=1}^{|w|} x_{w[i]}$.

Theorem 4 *Let $f_d(x_0, x_1) = \sum_{w \in \{0,1\}^d} x^w x^{w^R}$. Then*

1. $C(f_d) = \theta(d)$
2. $B(f_d) = 2^{\theta(d)}$
3. $F(f_d) = 2^{\theta(d)}$
4. $D(f_d) = \theta(d)$

Proof: (Sketch) The upper bound in all cases is obtained from the following linear size circuit for f_d:

$$f_d = x_0 * f_{d-1} * x_0 + x_1 * f_{d-1} * x_1$$

The lower bound in all cases is obtained by considering $M_d(f_d)$. This is easily seen to be a diagonal matrix of size 2^d. Using theorem 1 together with lemma 1 concludes the proof. $\square$

4.3 Monotone vs. non-monotone

Definition 4 *A circuit is called monotone if it does not use subtractions, and all real constants used are positive. An ABP is called monotone if all constants used as coefficients in the linear forms are positive. The monotone complexities of f are denoted by $C^+(f), D^+(f), F^+(f), B^+(f)$.*

Lemma 4 *All the results stated in lemma 1 also apply where $C(f), D(f), F(f), B(f)$ are all replaced with $C^+(f), D^+(f), F^+(f), B^+(f)$ resp.*

Proof: (sketch) All simulations used in the proof to lemma 1 preserve monotonicity. □

We will need the following definitions to get an analogue of theorem 1:

Definition 5 *For a function f of degree d, and $0 \leq k \leq d$, the k-ABP complexity of f, $B_k(f)$ is the minimum, over all ABPs that compute f, of the size of the k'th level of the ABP. The k-monotone-ABP complexity of f, $B_k^+(f)$ is the minimum, over all monotone ABPs that compute f, of the size of the k'th level of the ABP.*

Definition 6 *(Yannakakis [16]) For a real matrix M, the positive rank of M, $rank^+(M)$, is the minimum integer t such that there exist non-negative matrices A of dimension n by t and B of dimension t by m such that $M = AB$.*

Lemma 5 *For every homogeneous function f of degree d and all $0 \leq k \leq d$, $B_k^+(f) = rank^+(M_k(f))$. Also, $B^+(f) \geq \sum_{k=0}^{d} B_k^+(f)$.*

Proof:(Sketch) The only difference from the proof of theorem 1 is the fact that we cannot show that it is possible to obtain a single monotone ABP that achieves width $B_k^+(f)$ at all of its levels. What is obvious though is that for any k it is possible to obtain a monotone ABP that achieves width $B_k^+(f)$ at level k: Just compute the functions that correspond to the rows on A and to the columns of B in any manner (here AB is the decomposition of $M_k(f)$ giving $B_k^+(f)$). □

For a boolean function $b(x, y)$ denote by $C(b)$ the communication complexity of b and by $rank(b)$ the rank of the real matrix associated with it. Lovasz and Saks (see [7]) asked whether $C(b) = (\log rank(b))^{O(1)}$ for every boolean function b. It is known ([16], see also [7]) that the answer to this question is positive if and only if $\log(rank^+(M)) = \log(rank(M))^{O(1)}$ for every real matrix M with 0-1 entries.

Define an algebraic function to be simple if all of its coefficients are zero or one.

Theorem 5 *The following are equivalent:*

1. *For every boolean function b, $C(b) = (\log rank(b))^{O(1)}$.*
2. *For every simple algebraic function f and every k, $\log B_k^+(f) = (\log B_k(f))^{O(1)}$*

Proof: (Sketch) As we mentioned (1) is known to be equivalent to the fact that for every 0-1 matrix M: $\log rank^+(M) = (\log rank(M))^{O(1)}$. But in our notation every real matrix M corresponds to a function f such that $rank^+(M) = B_k^+(f)$ and $rank(M) = B_k(f)$. Moreover M is 0-1 if and only if f is simple. □

5 Commutative computation

In this section we use the lower bounds we have proved for non-commutative computation in order to prove lower bounds for commutative computation. As to avoid confusion between the two models we adopt the convention that $B(f)$, $D(f)$, $C(f)$, $B^+(f)$, etc. all refer to non-commutative complexity measures (as they have up to this point), while "circuit size", "circuit depth", etc. all refer to general commutative computation.

5.1 Monotone circuits

Recall the definition of *weakly equivalent* from definition 3, and notice that in this definition f or g may be also be commutative functions.

Lemma 6 *Let f be a function. If there exists a monotone formula of size s for f in the commutative case, then there exists a function f' which is weakly equivalent to f, s.t. $F^+(f') \leq s$.*

Proof: (sketch) The same formula for f, interpreted as a non-commutative formula, will compute a function which is weakly equivalent to f. □

Theorem 6 *([13, 14]): The permanent function requires $2^{\Omega(n)}$ monotone formula size, and $2^{\Omega(\sqrt{n})}$ monotone circuit size.*

Proof: The bound for formula size follows from theorem 2 and lemma 6. The bound for circuit size then follows from [8]. □

5.2 Depth three circuits

Definition 7 *Let $f(x_1 \ldots x_n) = \sum_t c_t \prod_{i \in S_t} x_i$ be a commutative function. The* strong non-commutative analogue *of f, f^* is defined to be a non-commutative function which has a monomial for any permutation of a monomial of f. I.e.*

$$f^*(x_1 \ldots x_n) = \sum_t \sum_\sigma c_t \prod_{i=1}^{|S_t|} x_{\sigma(i)}$$

Where σ ranges over all permutations on the elements of S_t.

We will consider here commutative circuits of the following form: A sum of products of linear functions. These will be called $\Sigma\Pi\Sigma$ circuits. A circuit is homogeneous if all of the linear functions used in it are homogeneous (i.e. have no constant term), and all products are of the same number of linear functions. The circuit is of degree D if all products are of at most D linear functions. It is clear that a homogeneous $\Sigma\Pi\Sigma$ circuit of degree D computes a function of degree D. Note that a $\Sigma\Pi\Sigma$ circuit can be converted to a $\Sigma\Pi\Sigma$ formula with only a polynomial penalty in size. It is known that non-homogeneous $\Sigma\Pi\Sigma$ circuits of degree n and polynomial size can compute any symmetric function of polynomial degree.

Lemma 7 *If there exists a $\Sigma\Pi\Sigma$ formula of size s and degree D for a homogeneous function f of degree d, then $F(f^*) \le O(sD!/(D-d)!)$.*

Proof: We build a non-commutative formula for f^* in two stages. In the first stage we convert the formula for f to a homogeneous one. This is done by partitioning each node in the original formula to its homogeneous components. For the addition gates in the lowest level we just separate and do not add the constants to the linear term. For each multiplication gate we need only recover the homogeneous component of degree d. This is done by summing over all subsets of size d of the inputs to the multiplication gate of the product of the d chosen linear terms by the other $D-d$ constant terms. This blows up the size by a factor of $\binom{D}{d}$. The top addition gate is then simply replaced by merging all the addition gates simulating the different multiplication gates.

Once we have a homogeneous formula for f we get one for f^* by having a multiplication gate for each permutation of inputs to a multiplication gate in the formula for f and adding up all these multiplication gates together. Note that for each monomial of f we now get all its permutations, thus obtaining a non-commutative formula for f^*. Also note that this only works since we have a single level of multiplication gates. This second stage increases the size by another factor of $d!$. $\square$

Let $\sigma_d(x_1 \ldots x_n)$ be the d'th elementary symmetric function. I.e.

$$\sigma_d(x_1 \ldots x_n) = \sum_{|S|=d} \prod_{i \in S} x_i$$

Lemma 8

$$B(\sigma_d^*) \ge \binom{n}{d/2}$$

Proof: (sketch) After removing duplicate rows and columns from $M_{d/2}(\sigma_d^*)$ one is left with the disjointness matrix of $d/2$ elements out of n. Kantor's lemma (see [1], chapter 6) states that this matrix has full rank. $\square$

We thus get:

Theorem 7 *If a $\Sigma\Pi\Sigma$ circuit of size s and degree D computes σ_d then $s \ge (n/(D^2 d))^{\Omega(d)}$. If a homogeneous $\Sigma\Pi\Sigma$ circuit of size s computes σ_d then $s \ge (n/d^3)^{\Omega(d)}$.*

Proof: (Sketch) Just put lemmas 7 and 8 together while noting that $B(f^*) \le F(f^*)$ and that circuit size and formula size agree to within a polynomial for $\Sigma\Pi\Sigma$ circuits. $\square$

6 Open problems

The major open problem is to use these techniques in order to prove lower bounds for general commutative computation in stronger ways than we have succeeded doing in section 5. There are also some questions about the non-commutative model:

- Is $D(f) = O(\log F(f))$?
- Prove a lower bound for non-commutative *circuit* size. (perhaps, as [4] attempted, for the determinant function.)
- Separate $F(f)$ from $B(f)$, or even $D(f)$ from $\log B(f)$. Equivalently prove a super-logarithmic lower bound for the depth required to multiply, non-commutatively, n matrices each of dimensions n by n.
- Separate, or prove the equality up to a polynomial, of monotone and non-monotone computation for some non-commutative complexity measure.

7 Acknowledgements

I have had very helpful discussions with M. Ben-Or, M. Karpinski, M. Krachmer, I. Newman, M. Safra, M. Sipser, and A. Wigderson, during various stages of this work.

References

[1] L. Babai and P. Frankl, *Linear algebra methods in combinatorics*, 1988.

[2] R.B. Boppana and M. Sipser, *The complexity of finite functions*, to appear in Handbook of computer science.

[3] J. von zur Gathen, *Algebraic complexity theory*, Ann. Rev. Comp. Sci. 3:317-47, 1988.

[4] L. Hyafil, *The power of commutativity*, 18th FOCS, 171-74, 1977.

[5] L. Hyafil, *On the parallel evaluation of multivariate polynomials*, 10th STOC, 193-195, 1978.

[6] M. Jerrum and M. Snir, *Some exact complexity results for straight-line computations over semirings*, Univ. of Edinburgh CRS-58-80, 1980.

[7] L. Lovasz, *Communication complexity: a survey*, 1989.

[8] G.L. Miller, V. Ramachandran, and E. Kaltofen, *Efficient parallel evaluation of straight line code and arithmetic circuits*, Siam J. Comp. 17, 1988.

[9] L. Valiant, *Negation can be exponentially powerful*, 1979.

[10] L. Valiant, S. Skyum, S. Berkowitz, and C. Rackoff, *Fast parallel computation of polynomials using few processors*, Siam J. Comp. 12:641-44, 1983.

[11] V. Strassen, *Vermeidung von divisionen*, J. Reine Angew Math., 264:184-202, 1973.

[12] C.P. Schnorr, *A lower bound on the number of additions in monotone computations*, TCS 2:305-315, 1976.

[13] E. Shamir and M. Snir, *Lower bounds on the number of multiplications and the number of additions in monotone computations*, IBM RC-6757, 1977.

[14] E. Shamir and M. Snir, *On the depth complexity of formulas*, Math. Systems theory, 13:301-322, 1980.

[15] S. Winograd, *On the number of multiplications necessary to compute certain functions*, Comm. on Pure and Appl. Math., 23:165-179, 1970.

[16] M. Yannakakis, *Expressing combinatorial optimization problems by linear programs*, 1988.