The Effect of Algebraic Structure on the Computational Complexity of Matrix Multiplication

Leslie Robert Kerr

Ph.D. Thesis
June 1970
70-75

Department of Computer Science
Cornell University
Ithaca, New York 14853

THE EFFECT OF ALGEBRAIC STRUCTURE ON THE COMPUTATIONAL COMPLEXITY OF MATRIX MULTIPLICATION

A Thesis

Presented to the Faculty of the Graduate School

of Cornell University for the Degree of

Doctor of Philosophy

by

Leslie Robert Kerr

June 1970

Leslie Kerr was born on August 1, 1944, in Ormstown, Quebec, Canada. He began studying Electrical Engineering at McGill University in 1961 and graduated With Distinction in 1966, receiving a B.Eng.(Hon.) degree. He then commenced graduate studies in Computer Science at Cornell University. Mr. Kerr is a member of the Association for Computing Machinery.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to Professor J. Hopcroft and Professor J. Hartmanis of the Department of Computer Science, Cornell University. Without the help and countless suggestions of Professor Hopcroft the completion of this thesis would not have been possible. The results of Chapter 3 expecially are due to him. I am indebted to Professor Hartmanis for his guidance during my years as a graduate student. It was his infectious enthusiasm that first attracted me to the study of automata theory and other topics in the theory of computation.

I am also grateful to Professor G. Salton and Professor R. Constable for reading early versions of the thesis and for their helpful suggestions.

This research has been supported in part by National Science Foundation Grant GJ-96.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

One of the more important goals of computer science is the development of better algorithms and computing devices. To accomplish this requires an improved understanding of functions and the algorithms used to compute them on various types of computers. Inherent in this investigation is the study of a function's computational complexity — intuitively a measure of the efficiency of the best possible algorithm for the function. Ideally one would like to know exactly what the complexity of a function is for all possible types of computing schemes. This would enable the design of computers and algorithms which optimize the efficiency of computation of any particular function or set of functions. One would then have the assurance that he was using the best possible algorithm and the best possible computer, within certain constraints such as the cost of the computer.

Although such an understanding of functions and their algorithms is clearly desirable, it is certainly a long way off. This thesis, however, attempts to show how this type of thinking can be applied to a particular problem, namely matrix multiplication. Since matrix multiplication appears to have a straightforward structure and is very easy to define, one would expect that its algorithms would have been carefully examined and that the problem would by now be well understood. It came as a great surprise, therefore, when Strassen recently pointed out [1] that the normal algorithm for matrix multiplication is by no means optimal, and that improved algorithms exist.

Strassen's method is based on a technique for multiplying two 2×2 matrices in seven multiplications rather than the eight that are normally used. This technique is described in detail in Chapter 3. Fortunately, the technique does not make use commutativity, so that it is valid even if the matrix elements are themselves matrices. Therefore, by letting the matrix elements be 2×2 matrices one can obtain a program for multiplying 4×4 matrices with just $7^2 = 49$ multiplications. By repeating this one obtains programs for multiplying $n \times n$ matrices with just $7^{\log_2 n}$ or $n^{\log_2 7}$ multiplications.

Although Strassen's method substantially increases the number of additions required for multiplying 2×2 matrices, for $n \times n$ matrices the number of additions is actually proportional to $n^{\log_2 7}$ (or roughly $n^{2.81}$). For large n ($n > 100$) the total number of arithmetic operations is therefore less than the number normally used which grows as n^3 . The question immediately arises whether or not Strassen's method can itself be improved. For example, if 2×2 matrices could be multiplied by using only six multiplications without assuming commutativity, then the bound for $n \times n$ matrices could be reduced to $n^{\log_2 6}$. These questions cannot be answered without a knowledge of the computational complexity of matrix multiplication.

The complexity of matrix multiplications is highly dependent upon the algebraic structure of the matrix elements. Strassen's method is valid only because $+$, $-$ and $\times$ possess certain properties — specifically, all the properties of a ring. On the other hand for matrix multiplication defined over two operations $\oplus$ and $\odot$, where $\oplus$ and $\odot$ satisfy no nontrivial properties (i.e. a free algebra), then no program is possible except the one which mimics the definition of matrix multiplication. In this case, therefore, n^3 multiplications and $n^2(n - 1)$ additions are both necessary and sufficient.

Although one ordinarily thinks of matrix multiplication in terms of matrices of numbers, many other useful problems can be formulated as matrix multiplications.

In many cases, the "addition" and "multiplication" operations do not satisfy all the axioms which Strassen's method requires, and one would like to know whether other forms of improved algorithms are possible. For example, the shortest path matrix of a graph can be calculated using matrix multiplications, where the "addition" operation is <u>min</u> and the "multiplication" operation is $+$. In this case all the ring axioms are satisfied except that there is no additive inverse; hence Strassen's method cannot be used for this particular algebra. In fact, it is shown later in the thesis that there do not exist any improved algorithms for <u>min</u> and $+$.

The model of computation used in this thesis is a reasonably computer-like one called <u>fixed programs</u>. Fixed programs, described more formally in Chapter 2, are essentially computer programs with no conditional transfer or looping instructions. Previous researchers have obtained complexity results for some specific functions relative to fixed programs. Pan [2] proved that n additions or subtractions and n multiplications or divisions are necessary to evaluate an nth degree polynomial with arbitrary coefficients. His method of proof is based on the observation that if a program evaluates a polynomial in n-space, then it must also evaluate it on a rational surface in n-space. This rational surface is carefully chosen so that one of the multiplications of the program is identically equal to zero over the rational surface. The dimension or number of degrees of freedom of this surface is $n - 1$. The conclusion is reached that if k multiplications are required to compute the matrix multiplication problem on the surface, then $k + 1$ are required to compute it in n-space. This process is repeated for surfaces with fewer and fewer degrees of freedom, until finally the surface is reduced to a point. Since n multiplications were removed from the original program, at least n multiplications must have been originally included. A similar argument can be used to prove that at least n additions or subtractions are required. Hence one can conclude that Horner's method for evaluating polynomials minimizes the number of arithmetic operations for fixed programs, because it uses exactly n additions and n multiplications.

Winograd [3] used essentially the same method to prove that the multiplication of an arbitrary $m \times n$ matrix by an arbitrary n-vector requires at least mn multiplications or divisions. This result and Pan's result for polynomials actually drop out as special cases of a more general theorem by Winograd which is described later in Chapter 2.

Using Pan's method of eliminating multiplications by reducing the dimension of the space can result in bounds of at most p, where p is the total number of arguments of the function. The highest bound to be expected for the multiplication of two $n \times n$ matrices would therefore be $2n^2$, To get better results some new techniques must be developed, and much of this thesis is devoted to the development of these techniques. In particular, matrix multiplication is very complex and difficult to analyze until one takes advantage of the large degree of symmetry inherent in the problem. A scheme will be described which systematically takes into account the symmetry and reduces the problem to much simpler ones.

In the next chapter a precise definition of fixed programs is described and then a type of complexity measure is defined for classes of functions relative to fixed programs. The succeeding chapters deal with various algebraic structures and the establishment of bounds on the complexity of the matrix multiplication problem for each structure. In Chapter 3 a generalization of Strassen's method is presented which is valid whenever the matrix elements satisfy all the axioms of a noncommutative ring. In Chapter 4 a lower bound on the complexity for noncommutative rings is obtained which shows some special cases of this algorithm to be optimal with respect to the number of multiplications used. In Chapter 5 the complexity for commutative rings is studied, and it is shown that in this case programs exist which require fewer multiplications than Strassen's method. In Chapter 6 it is shown that if either the additive inverse or the distributivity axioms are removed from the ring axioms, then the complexity of matrix multiplication for the resulting structure is n^3, and hence no improved algorithms exist. Chapter 7 contains some concluding remarks and suggestions for further research.

CHAPTER 2

PRELIMINARIES

The model of computation used in this thesis is one which we call fixed programs — identical to the model used by Pan [2] and Winograd [3]. Fixed programs, to be defined more precisely later in this chapter, are essentially computer programs with instruction repertoires restricted to arithmetic operations, i.e. no conditional transfer or looping instructions. One might argue that in limiting ourselves to fixed programs we are excessively restricting the class of functions which can be computed. In a sense this is true, since with fixed programs over $+$, $-$, $\times$ and $\div$ we can compute only rational functions, excluding even such simple functions as exponential and factorial.

To counter this argument, however, one observes that in real computer programs the full potential of the looping capability is rarely exploited. In fact, the looping which is present in most programs is usually no more than is required to simulate a class of fixed programs. A typical program for evaluating polynomials, for example, when presented with a polynomial of degree n would simply loop n times in order to simulate the fixed program for evaluating nth degree polynomials.

Perhaps the best argument for fixed programs is the intuitive feeling that in the computation of rational functions, such as matrix multiplication, the use of conditional transfer and looping instructions cannot reduce the total number of operations required.

Fixed programs can be described more formally as follows:

<u>Definition</u>: A <u>fixed program</u>, or simply a <u>program</u>, is a finite sequence of instructions of the form

$$t_i \leftarrow \alpha \; 0 \; \beta \qquad 1 \leq i \leq n$$

where α and β are either input variables of the program, constants, or previous t_j's (i.e. $j < i$) and 0 is some operation in the instruction repertoire available to the program. The program is executed by assigning to each t_i in turn the value of $\alpha \; 0 \; \beta$. A program is said to compute a function $f(x_1,...,x_k)$ if the value of one of the t_i's is equal to $f(x_1,...,x_k)$ for all possible values of the arguments $x_1,...,x_k$. A program is said to compute a set of functions $F = \{f_1,...,f_m\}$ if there is a set of t_i's, $T = \{t_{j_1},...,t_{j_m}\}$, such that the value of t_{j_i} is equal to f_i, $1 \leq i \leq m$.

An example of a fixed program is the following one which computes the polynomial $ax^2 + bx + c$ using Horner's method.

$$t_1 \leftarrow a \times x$$
$$t_2 \leftarrow t_1 + b$$
$$t_3 \leftarrow t_2 \times x$$
$$t_4 \leftarrow t_3 + c$$

This program computes $ax^2 + bx + c$ because the value of t_4 is $ax^2 + bx + c$.

Matrix multiplication is usually defined in terms of ordinary addition and multiplication, the elements of the matrices being generally either integers or real numbers. Since we wish to study the complexity of matrix multiplication as a function of the algebraic structure of the matrix elements, we now present a slightly more general definition of the matrix multiplication problem. Let $\mathcal{Q} = \langle S, \oplus, \odot \rangle$ be an algebra with two operations. Then the matrix multiplication problem for $\mathcal{Q}$ can be defined as follows:

<u>Definition</u>: Let $A = \{a_{ij} | 1 \leq i \leq m , 1 \leq j \leq p\}$ be an $m \times p$ matrix and let $X = \{x_{ij} | 1 \leq i \leq p , 1 \leq j \leq n\}$ be a $p \times n$ matrix. Then the product $M = A \times X$ is

defined to be an $m \times n$ matrix

$M = \{M_{ij} | 1 \leq i \leq m \ , \ 1 \leq j \leq n\}$ where

$$M_{ij} = a_{i1} \odot x_{1j} \oplus \ldots \oplus a_{ip} \odot x_{pj} \ .$$

M will usually be denoted by $M(m,p,n)$ to indicate the dimensions of A and X .

The algebraic structures considered in this thesis satisfy one or more of the following axioms:

A1) Addition is associative.

$$(a \oplus b) \oplus c = a \oplus (b \oplus c)$$

A2) Addition is commutative.

$$a \oplus b = b \oplus a$$

A3) Additive identity.

$$(\exists 0)(\forall a) \quad a \oplus 0 = a$$

A4) Additive inverse.

$$(\forall a)(\exists a') \quad a \oplus a' = 0$$

a' is usually denoted by $-a$,

M1) Multiplication is associative.

$$(a \odot b) \odot c = a \odot (b \odot c)$$

M2) Multiplication is distributive (over addition).

$$a \odot (b \oplus c) = a \odot b \oplus a \odot c$$

M3) Multiplication is commutative.

$$a \odot b = b \odot a$$

M4) Multiplicative identity.

$$(\exists 1)(\forall a) \quad 1 \odot a = a$$

These axioms are satisfied by the following algebraic types:

<u>Noncummutative ring</u>: A1, A2, A3, A4, M1 and M2 .
<u>Commutative ring</u>: A1, A2, A3, A4, M1, M2 and M3 .
<u>Ring with unity</u>: A1, A2, A3, A4, M1, M2, M3 and M4 .
"<u>Ring without minus</u>": A1, A2, A3, M1, M2, M3 and M4 .

Note that while this structure is not a ring, it satisfies all the axioms of a ring except the additive inverse axiom.

"<u>Nondistributive ring</u>": A1, A2, A3, A4 and M1 .

While this structure is not a ring, it satisfies all the axioms of a ring except the distributivity axiom.

It is possible to say that a program P computes an expression E for an algebra $\mathcal{A} = \langle S, \oplus, \odot \rangle$.

By this we mean that if the addition and multiplication instructions in P are replaced by the $\oplus$ and $\odot$ operations of $\mathcal{a}$ (and minus is replaced by $\ominus$ if an additive inverse exists for $\mathcal{a}$) then the output of the program is equal to the expression E for any set of arguments in S . Thus the sample program given earlier in the chapter computes $a \odot x \odot x \oplus b \odot x \oplus c$ for the real numbers under + and - , for sets under $\cup$ and $\cap$, in face for any algebra in which $\odot$ distributes over $\oplus$.

We now define precisely the measure of computational complexity which is used in this thesis.

<u>Definition</u>: The complexity of a set of expressions E for some algebra $\mathcal{a} = \langle S, \oplus , \odot \rangle$ is the least number k such that there is some fixed program P which computes E for $\mathcal{a}$ and P has k instructions of the type $t_i \leftarrow \alpha \odot \beta$ (i.e. P uses k multiplications).

Note that since all our programs are finite there do exist optimal programs. This does not contradict the results of Blum [4] because fixed programs cannot compute all the computable functions, and therefore Blum's axioms are not satisfied.

This definition can be extended to classes of algebras (e.g. the class of all noncommutative rings) as follows:

<u>Definition</u>: The complexity of a set of expressions E for a class of algebras $\mathbf{A}$ is the least number k such that there is a fixed program P which computes E for <u>all</u> algebras in $\mathbf{A}$ and P has k multiplication instructions.

It is clear from the definition of the matrix multiplication problem that the complexity of $M(m,p,n)$ for any algebra $<S, \oplus, \odot>$ can be no more than mpn, since the definition itself is essentially a fixed program having just the mpn multiplications $a_{ik} \odot x_{kj}$. It is shown in later chapters, however, that the complexity of $M(m,p,n)$ for some algebras, such as all the noncommutative rings, is substantially less than mpn. We are now able to state a simple, but fundamental, theorem on the complexity of functions for classes of algebras.

<u>Theorem 2.1</u>: Let $\mathbf{A}_1$ and $\mathbf{A}_2$ be two classes of algebras such that $\mathbf{A}_1 \subseteq \mathbf{A}_2$. Then the complexity of a set of expressions E for $\mathbf{A}_1$ is less than or equal to the complexity of E for $\mathbf{A}_2$.

<u>Proof</u>: Let the complexity of E for $\mathbf{A}_2$ be k. Then there must be some program P which computes E for each algebra in $\mathbf{A}_2$ and has k multiplication instructions. Since each algebra in $\mathbf{A}_1$ is also in $\mathbf{A}_2$,

P computes E for $\mathbf{A}_1$. Therefore the complexity of E for $\mathbf{A}_1$ can be no more than k . $\square$

It is instructive at this point to examine a theorem due to Winograd [3]. This is one of the major results obtained in an effort to prove programs optimal for rational functions, and will be referred to several times throughout the thesis. The theorem is concerned with the number of multiplications or divisions required to compute linear functions of the variables $x_1,\ldots,x_n$. Let F be any field and let G be a subfield of F . Let Φ be a $t \times n$ matrix and ϕ a t-vector, the elements of which are all in the field F . We will use $\Phi_1,\ldots,\Phi_n$ to denote the columns of Φ . Let $\underline{x}$ denote the n-vector $(x_1,\ldots,x_n)$, and G^t denote the Cartesian product $G \times G \times \ldots \times G$. The theorem provides a lower bound on the number of multiplications or divisions (abbreviated m/d) required to compute the t elements $\Phi\underline{x} + \phi$.

<u>Theorem 2.2</u> (Winograd): Let P be a program computing $\Phi\underline{x} + \phi$. If there are u vectors in $\{\Phi_1,\ldots,\Phi_n\}$ such that no nontrivial linear combination of them (over G) is in G^t , Then P has at least u multiplications or divisions.

<u>Proof</u>: We will use the phrase "m/d which is counted" to denote a multiplication $t_i \leftarrow \alpha \cdot \beta$ such that neither α nor β is in G, or a division $t_i \leftarrow \alpha \div \beta$ such that β is not in G.

By induction on q, it is easily verified that if the first q instructions of P do not involve an m/d which is counted, then for $1 \leq i \leq q$ the value of t_i can be expressed as

$$\sum_{j=1}^{n} g_j x_j + f$$

where the g_j's are in G and f is in F.

We will prove the theorem by induction on u. If $u = 1$ there exists i_o and j_o such that $\Phi_{i_o j_o}$ is not in G. Assume that no m/d which is counted appears in P. Since P computes $\Phi\underline{x} + \phi$, for some k the value of t_k is $\sum_{j=1}^{n} \Phi_{i_o,j} + \phi_{i_o}$. But since P has no m/d which is counted there exist g_j's in G and f in F such that the value of t_k is $\sum_{j=1}^{n} g_j x_j + f$, and therefore $g_{j_o} = \Phi_{i_o,j_o}$ contradicting the assumption that Φ_{i_o,j_o} is not in G.

Now assume that the theorem holds for u (and all t and n), and let P be a program computing $\Phi\underline{x} + \phi$ such that P minimizes the number of m/d which are counted, and there are at least $u + 1$ vectors in $\{\phi_1, \ldots, \phi_n\}$ such that no nontrivial linear combination of them (over G) is in G^t. Let t_k be the first m/d in P which is counted. Then either

$$t_k = (\sum_{i=1}^{n} g_i x_i + f) \cdot (\sum_{i=1}^{n} h_i x_i + f')$$

or

$$t_k = (\sum_{i=1}^{n} g_i x_i + f) \div (\sum_{i=1}^{n} h_i x_i + f')$$

where the g_i's and h_i's are in G and f and f' are in F. Furthermore, either one of the g_i's or one of the h_i's is not 0; otherwise t_k is in G and no m/d would be required in a minimal program. Assume without loss of generality that $h_n \neq 0$, and since multiplication by h_n or h_n^{-1} is not counted we can further assume that $h_n = 1$.

Let g be an element of G such that if we substitute $g - f' - \sum_{i=1}^{n-1} h_i x_i$ for x_n the resulting program

P' is such that t'_k is total. Substituting

$g - f' - \sum_{i=1}^{n-1} h_i x_i$ for x_n we obtain a program which computes $\Phi'\underline{x}' + \phi'$ where $\Phi'_j = \Phi_j - h_i\Phi_n$ $1 \leq j \leq n - 1$, $\phi' = \phi + (g - f')\Phi_n$, and $\underline{x}' = (x_1,\ldots,x_{n-1})$. The number of m/d which are counted in P' is at least one fewer than in P since step k is not an m/d which is counted in P' , and the program computing $g - f' - \sum_{i=1}^{n-1} h_i x_i$ has no m/d which is counted. But there exist at least u vectors in $\{\Phi'_1,\ldots,\Phi'_{n-1}\}$ such that no nontrivial linear combination of them (over G) is in G^t , therefore by the induction hypothesis P' has at least u m/d which are counted. Hence P has at least $u + 1$ m/d which are counted. □

<u>Corollary 1</u> (Pan): No prog[illegible] for evaluating $\sum_{i=0}^{n} x_i y^i$ requires fewer than n m/d, and therefore Horner's rule minimizes the number of m/d in computing a polynomial.

<u>Proof</u>: Let $F = G(y)$. Then $\sum_{i=0}^{n} x_i y^i = \Phi\underline{x}$ where Φ is the $1 \times (n+1)$ matrix $[1\ y \ldots y^n]$. Since

$\{y,\ldots,y^n\}$ are linearly independent over G, by Theorem 2.2 at least n m/d are required. $\square$

<u>Corollary 2</u> (Winograd): Let X be a $p \times q$ matrix, and let $\underline{y}$ be a q-vector. No program for computing $X\underline{y}$ requires fewer than pq m/d, and therefore the ordinary way for computing $X\underline{y}$ minimizes the number of m/d.

<u>Proof</u>: Let $F = G(y_1,\ldots,y_q)$ and let Φ be the $p \times pq$ matrix where

$$\Phi_{ij} = \begin{cases} y_k & \text{if } j = iq + k \quad 1 \le k \le q \\ 0 & \text{otherwise} \end{cases}$$

and let $\underline{x}$ be the column vector $(x_{11},\ldots,x_{1q},x_{21},\ldots,x_{2q},\ldots,x_{pq})$. Clearly $X\underline{y} = \Phi\underline{x}$ and since no nontrivial linear combination (over G) of the columns of Φ is in G^p the corollary is obtained by applying Theorem 2.2 $\square$

CHAPTER 3

NONCOMMUTATIVE RINGS – UPPER BOUND

In this chapter an upper bound is obtained on the complexity of the matrix multiplication problem for non-commutative rings. Specifically, a program is described which computes the matrix multiplication problem $M(m,2,n)$ using just $\left\lceil \frac{3mn + \max(m,n)}{2} \right\rceil$ multiplications for all algebras which satisfy the following axioms:

Addition: associative, commutative, identity, inverse;

Multiplication: associative, distributive.

The program which is presented is an extension of a method due to Strassen [1] for computing $M(n,n,n)$ with just $n^{\log_2 7}$ or approximately $n^{2.81}$ multiplications. Let A be an $m \times 2$ matrix with elements a_{ij} , $1 \leq i \leq m$, $1 \leq j \leq 2$ and let X be a $2 \times n$ matrix with elements x_{ij} , $1 \leq i \leq 2$, $1 \leq j \leq n$. We shall make use of the following notation. If $Q(a_i, x_k) = a_{i2}(x_{1k} + x_{2k})$, where Q represents one of the letters $A, B, \ldots, G$, then we write $Q(a_{i+j}, x_k)$ for

$a_{i+j,2}(x_{1k}+x_{2k})$ and $Q(a_i+a_j,x_k)$ for $(a_{i2}+a_{j2})(x_{1k}+x_{2k})$.

Let

$$\begin{aligned}
A(a_i,x_k) &= a_{i2}(x_{1k} + x_{2k}) \\
B(a_i,x_k) &= (a_{i1} - a_{i2})x_{1k} \\
C(a_j,x_\ell) &= (a_{j1} - a_{j2})x_{2\ell} \\
D(a_j,x_\ell) &= a_{j1}(x_{1\ell} + x_{2\ell}) \\
E(a_i+a_j,x_k+x_\ell) &= (a_{i2} + a_{j2})(x_{2k} + x_{2\ell}) \\
F(a_i+a_j,x_k+x_\ell) &= (a_{i1} + a_{j1})(a_{1k} + x_{1\ell}) \\
G(a_i,a_j,x_k,x_\ell) &= (a_{j1} + a_{i2})(a_{1k} - x_{2\ell}) .
\end{aligned}$$

Strassen's algorithm computes

$$\begin{aligned}
y_{ik} &= a_{i1}x_{1k} + a_{i2}x_{2k} , \\
y_{i\ell} &= a_{i1}x_{1\ell} + a_{i2}x_{2\ell} , \\
y_{jk} &= a_{j1}x_{1k} + a_{j2}x_{2k}
\end{aligned}$$

and

$$y_{j\ell} = a_{j1}x_{1\ell} + a_{j2}x_{2\ell}$$

as

$$\begin{aligned}
y_{ik} &= A(a_i,x_k) + B(a_i,x_k) \\
y_{i\ell} &= -B(a_i,x_k)-D(a_j,x_\ell)+F(a_i+a_j,x_k+x_\ell)-G(a_i,a_j,x_k,x_\ell) \\
y_{jk} &= -A(a_i,x_k)+C(a_j,x_\ell)+E(a_i+a_j,x_k+x_\ell)+G(a_i,a_j,x_k,x_\ell) \\
y_{j\ell} &= -C(a_j,x_\ell) + D(a_j,x_\ell) .
\end{aligned}$$

Note that $y_{ik} + y_{i\ell}$, $y_{jk} + y_{j\ell}$, $y_{ik} + y_{jk}$, $y_{i\ell} + y_{j\ell}$ and $y_{ik} + y_{i\ell} + y_{jk} + y_{j\ell}$ can be computed with no additional multiplications. The formulae are summarized below:

$$y_{ik}+y_{i\ell} = A(a_i,x_k)-D(a_j,x_\ell)+F(a_i+a_j,x_k+x_\ell)-G(a_i,a_j,x_k,x_\ell)$$
$$y_{jk}+y_{j\ell} = -A(a_i,x_k)+D(a_j,x_\ell)+E(a_i+a_j,x_k+x_\ell)+G(a_i,a_j,x_k,x_\ell)$$
$$y_{ik}+y_{jk} = B(a_i,x_k)+C(a_j,x_\ell)+E(a_i+a_j,x_k+x_\ell)+G(a_i,a_j,x_k,x_\ell)$$
$$y_{i\ell}+y_{j\ell} = -B(a_i,x_k)-C(a_j,x_\ell)+F(a_i+a_j,x_k+x_\ell)-G(a_i,a_j,x_k,x_\ell)$$
$$y_{ik}+y_{i\ell}+y_{jk}+y_{j\ell} = E(a_i+a_j,x_k+x_\ell)+F(a_i+a_j,x_k+x_\ell)$$

The essence of the algorithm to be given is to compute groups of four elements using Strassen's method and to share as many multiplications between groups as possible. For example, the 3×2 by 2×3 case would be computed as:

$$y_{11} = A(a_1,x_1) + B(a_1,x_1)$$
$$y_{12} = -B(a_1,x_1) - D(a_2,x_2) + F(a_1+a_2,x_1+x_2) - G(a_1,a_2,x_1,x_2)$$
$$y_{13} = A(a_1,x_1) - D(-a_1+a_3,-x_1+x_3) + F(a_3,x_3) - G(a_1,-a_1+a_3,x_1,-x_1+x_3)$$
$$y_{21} = -A(a_1,x_1) + C(a_2,x_2) + E(a_1+a_2,x_1+x_2) + G(a_1,a_2,x_1,x_2)$$
$$y_{22} = -C(a_2,x_2) + D(a_2,x_2)$$
$$y_{23} = -A(-a_2+a_3,-x_2+x_3) + D(a_2,x_2) + E(a_3,x_3) + G(-a_2+a_3,a_2,-x_2+x_3,x_2)$$

$$y_{31} = B(a_1,x_1) + D(-a_1+a_3,-x_1+x_3) + E(a_3,x_3) + G(a_1,-a_1+a_3,x_1,-x_1+x_3)$$

$$y_{32} = -B(-a_2+a_3,-x_2+x_3) - C(a_2,x_2) + F(a_3,x_3) - G(-a_2+a_3,a_2,-x_2+x_3,x_2)$$

$$y_{33} = E(a_3,x_3) + F(a_3,x_3) \ .$$

Here we have computed y_{11} , y_{12} , y_{21} and y_{22} as a group. Similarly y_{22} , y_{23} , y_{32} and y_{33} and y_{11} , y_{13} , y_{31} and y_{33} . Note that two multiplications are common to the first and second groups and two multiplications are common to the first and third groups. In all 15 multiplications are used.

Before giving the general construction we proceed with several technical lemmas.

Lemma 3.1: Let $a_1,a_2,\ldots,a_m$ be indeterminates. For $n \geq m$ there exist linear functionals $\ell_1,\ell_2,\ldots,\ell_{n-m}$ such that for any k , $a_k,a_{k+1},\ldots,a_{k+m-1}$ are nondependent where $a_{m+i} = \ell_i(a_1,a_2,\ldots,a_m)$, $1 \leq i \leq n-m$.[†]

[†]For notational convenience we interpret $a_k,a_{k+1},\ldots,a_{k+m}$ for $k+m > n$ as $a_k,a_{k+1},\ldots,a_n,a_1,a_2,\ldots,a_{k+m-n}$. We shall adhere to this convention throughout the paper. The indeterminates $a_1,a_2,\ldots,a_n$ are nondependent if for scalars c_i , $1 \leq i \leq k$, $\sum_{i=1}^{k} c_i a_i = 0$ implies $c_i = 0$, $1 \leq i \leq k$.

Proof: We can interpret the sequence $a_1, a_2, \ldots, a_n$ as an $n \times m$ matrix L where the ith row represents a linear functional expressing a_i in terms of $a_1, a_2, \ldots, a_m$. Thus we need only show how to construct an $n \times m$ matrix in which

1) the first m rows form the identity matrix, and
2) each submatrix consisting of m consecutive rows of L has a non-zero determinant. (Here rows n and 1 are also considered to be consecutive.)

This is equivalent to constructing an $n \times m$ matrix such that

1) the first m rows form the identity matrix
2) $$\begin{vmatrix} a_{i1} & \cdots & a_{im} \\ \vdots & & \vdots \\ a_{i+m-1,1} & \cdots & a_{i+m-1,m} \end{vmatrix} \neq 0 \ , \ 1 \leq i \leq n-m+1$$
3) $$\begin{vmatrix} a_{n-i,m-i} & \cdots & a_{n-i,m} \\ \vdots & & \vdots \\ a_{n,m-i} & \cdots & a_{nm} \end{vmatrix} \neq 0 \ , \ 0 \leq i \leq m-1 \ .$$

Clearly for $n = m$, the identity matrix satisfies the above three conditions. Assume we have an $n \times m$ matrix satisfying the above conditions. Then we can construct an $(n+1) \times m$ matrix satisfying the above conditions since for each i the determinants

$$D(i) = \begin{vmatrix} a_{n+1-i,m-i} & \cdots & a_{n+1-i,m} \\ \vdots & & \vdots \\ a_{n+1,m-i} & \cdots & a_{n+1,m} \end{vmatrix} \qquad 0 \le i \le m-1$$

depend on the value of $a_{n+1,m-i}$. Thus starting with $i = 0$ for each i, $0 < i \le m-1$, select $a_{n+1,m-i}$ so that $D(i) \neq 0$. □

<u>Lemma 3.2</u>: If y_{ii} and y_{jj} are each computed by one of the following three methods

1) $y_{tt} = A(a_t,x_t) + B(a_t,x_t)$
2) $y_{tt} = -C(a_t,x_t) + D(a_t,x_t)$
3) $y_{tt} = E(a_t,x_t) + F(a_t,x_t)$

but not the same method, then y_{ij} and y_{ji} can be computed with three additional multiplications.

<u>Proof</u>: If y_{ii} is computed by (1) and y_{jj} by (2), then $y_{ij} = -B(a_i,x_i) - D(a_j,x_j) + F(a_i+a_j,x_i+x_j) - G(a_i,a_j,x_i,x_j)$ and $y_{ji} = -A(a_i,x_i) + C(a_j,x_j) + E(a_i+a_j,x_i+x_j) + G(a_i,a_j,x_i,x_j)$.

If y_{ii} is computed by (1) and y_{jj} by (3), then $y_{ij} = A(a_i,x_i) - D(-a_i+a_j,-x_i+x_j) + F(a_j,x_j) -G(a_i,-a_i+a_j,x_i,-x_i+x_j)$ and $y_{ji} = B(a_i,x_i)$

$+ C(-a_i+a_j,-x_i+x_j) + E(a_j,x_j) + G(a_i,-a_i+a_j,x_i,-x_i+x_j)$.

If y_{ii} is computed by (2) and y_{jj} by (3), then $y_{ij} = -A(-a_i+a_j,-x_i+x_j) + D(a_i,x_i) + E(a_j,x_j)$ $+G(-a_i+a_j,a_i,-x_i+x_j,x_i)$ and $y_{ji} = -B(-a_i+a_j,-x_i+x_j)$ $- C(a_i,x_i) + F(a_j,x_j) - G(-a_i+a_j,a_i,-x_i+x_j,x_i)$.

The other three cases are obtained vy interchanging the roles of i and j . □

To simplify the next lemma, we define two constants ℓ and m . Let n and k be positive integers where $2k+2 \leq n < 4k+4$. If $2k+2 \leq n \leq 3k+3$, then $\ell = 2k+3$ and $m = n - (2k+2)$ or $n - (2k+3)$, whichever is even. If $3k+3 < n < 4k+4$, then $\ell = n - k$ and $m = 4k+4 - n$ or $4k+3 - n$, whichever is even.

<u>Lemma 3.3</u>: A sequence $i_1,i_2,\ldots,i_n$ where $i_j = 1$, 2 or 3 for $1 \leq j \leq n$ can be constructed satisfying the properties:

(1) $i_j \neq i_{j+1} \quad 1 \leq j \leq n$

where i_{n+1} is interpreted to be i_1 .

(2) $$\left.\begin{array}{l} i_j \neq i_{j-k} \\ i_{j+1} \neq i_{j+k+1} \end{array}\right\} j = \ell,\ell+2,\ldots,\ell+m-2$$

(3) either $i_j \neq i_{j+k+1}$, or

$$i_{j+1} \neq i_{j-k} \quad \text{for} \quad j = \ell, \ell+2, \ldots, \ell+m-2 \ .$$

Proof: If $m < 2$, the lemma is trivial. Thus assume $m \geq 2$. Let $i_\ell = i_{\ell+2} = \ldots = i_{\ell+m-2} = 1$, $i_{\ell+1} = i_{\ell+3} = \ldots = i_{\ell+m-1} = 2$, $i_{\ell-k} = i_{\ell+2-k} = \ldots = i_{\ell+m-2-k} = 2$ and $i_{\ell+1+k} = i_{\ell+3+k} = \ldots = i_{\ell+m-1+k} = 3$. For $m \geq 2$, $\ell-k \leq \ell+m-2-k < \ell < \ell+m-1 < \ell+1+k \leq \ell+m-1+k$ and $\ell+m-1+k-n < \ell-k$. Thus, so far no conditions are violated. The remaining entries in the sequence are filled in so as to satisfy condition 1. □

Theorem 3.1: Let A be an $m \times 2$ matrix whose elements are indeterminates a_{ij} , $1 \leq i \leq m$, $1 \leq j \leq 2$ and let X be a $2 \times n$ matrix whose elements are indeterminates x_{jk} , $1 \leq j \leq 2$, $1 \leq k \leq n$. Then $\left\lceil \frac{3mn + \max(m,n)}{2} \right\rceil$ multiplications are sufficient to compute $A \times X$ without making use of commutativity.

Proof: Without loss of generality assume $m \leq n$. (Otherwise let $A' = X^T$ and $X' = A^T$ and compute $A' \times X'$.) Without loss of generality assume $n < 2m$. (Otherwise write $n = km + n'$, $k \geq 1$, $m \leq n' < 2m$ and decompose A into k $n \times 2$ matrices and an $n' \times 2$

matrix. Computing the produce of X with each submatrix requires

$$k\left\lceil\frac{3m^2+m}{2}\right\rceil + \left\lceil\frac{3mn'+n'}{2}\right\rceil = \left\lceil\frac{3mn+n}{2}\right\rceil .^{\dagger}$$

Let $a_1, a_2, \ldots, a_m$ be indeterminates. Let $\ell_1, \ell_2, \ldots, \ell_{n-m}$ be linear functionals of $a_1, a_2, \ldots, a_m$ such that the sequence $a_1, a_2, \ldots, a_m$, $a_{m+1}, \ldots, a_n$ where $a_{m+i} = \ell_i(a_1, a_2, \ldots, a_m)$ $1 \le i \le n-m$ satisfies Lemma 3.1. Augment the matrix A with $n - m$ rows where $a_{ij} = \ell_{i-m}(a_{1j}, a_{2j}, \ldots, a_{mj})$, $m+1 \le i \le n$, $1 \le j \le 2$. Call the augmented matrix $\bar{A}$. It suffices to compute m consecutive elements in each column of AX. We consider t[illegible] cases. Both cases will make use of the following b[illegible] construction.

<u>Basic Construction</u> [illegible]t k be an integer between 0 and $(n-1)/2$. The[illegible]ements y_{ij}, $1 \le j \le n$, $j \le k \le i \le j+k$ c[illegible] be computed in $3kn + 2n$ multipli-cations as follows. For each i compute y_{ii} according to one of the following three methods.

(1) $y_{ii} = A(a_i, x_i) + B(a_i, x_i)$

(2) $y_{ii} = -C(a_i, x_i) + D(a_i, x_i)$

---

† Note $\left\lceil\frac{3m^2+m}{2}\right\rceil = \frac{3m^2+m}{2}$.

(3) $y_{ii} = E(a_i,x_i) + F(a_i,x_i)$

subject to the restriction that if y_{ii} is computed according to method j, then $y_{i+1,i+1}$ is not computed by method j. Next, each needed y_{ij} and y_{ji}, such that y_{ii} and y_{jj} are computed by distinct methods, are computed according to Lemma 3.2 at a cost of three additional multiplications per pair of elements. Finally those y_{ij} and y_{ji}, such that y_{ii} and y_{jj} are computed by the same method, are obtained as follows. Several cases exist. We treat only the case where y_{ii} is computed as $A(a_i,x_i) + B(a_i,x_i)$, y_{jj} is computed as $A(a_j,x_j) + B(a_j,x_j)$ and $y_{i+1,i+1}$ is computed as $-C(a_{i+1},x_{i+1}) + D(a_{i+1},x_{i+1})$. In this case the multiplications $E(a_i+a_{i+1},x_i+x_{i+1})$ and $F(a_i+a_{i+1},x_i+x_{i+1})$ are already used in the computation of $y_{i,i+1}$ and $y_{i+1,i}$. Thus we can compute

$$\begin{aligned}
y_{ij} + y_{i+1,j} = B(a_j,x_j) &+ C(-a_j+a_i+a_{i+1},-x_j+x_i+x_{i+1}) \\
&+ E(a_i+a_{i+1},x_i+x_{i+1}) \\
&+ G(a_j,-a_j+a_i+a_{i+1},x_j,-x_j+x_i+x_{i+1}) \\
y_{ji} + y_{j,i+1} = A(a_j,x_j) &- D(-a_j+a_i+a_{i+1},-x_j+x_i+x_{i+1}) \\
&+ F(a_i+a_{i+1},x_i+x_{i+1}) \\
&- G(a_j,-a_j+a_i+a_{i+1},x_j,-x_j+x_i+x_{i+1})
\end{aligned}$$

requiring only the three additional multiplications C, D and G. Now $y_{i+1,j}$ and $y_{j,i+1}$ are already computed since y_{ii} and $y_{i+1,i+1}$ are computed by distinct methods and hence y_{jj} and $y_{i+1,i+1}$ are computed by distinct methods. (If $y_{i+1,j}$ and $y_{j,i+1}$ are not computed because i is not in the range $j-k \leq i \leq j+k$, then reverse i and j.) Thus we obtain

$y_{ij} = y_{ij} + y_{i+1,j} - y_{i+1,j}$ and

$y_{ji} = y_{ji} + y_{j,i+1} - y_{j,i+1}$. The total cost is $2n + 3kn$ as was to be shown.

We now return to computing m consecutive elements in each column of $\overline{A}X$. We consider two cases.

<u>Case 1</u>: $m = 2k + 1$, $k = 0,1,2\ldots$.

Let $k = (m-1)/2$. For each i we compute $y_{i-k,i}, y_{i-k+1,i}, \ldots, y_{i+k,i}$ by the basic construction computing the diagonal elements by

$$y_{ii} = \begin{cases} A(a_i,x_i) + B(a_i,x_i) & i \text{ odd} \quad i \neq n \\ -C(a_i,x_i) + D(a_i,x_i) & i \text{ even} \\ E(a_i,x_i) + F(a_i,x_i) & i \text{ odd} \quad i = n \end{cases}$$

This requires $(3mn+n)/2$ multiplications which is always an integer and hence equal to $\lceil(3mn+n)/2\rceil$.

<u>Case 2</u>: $m = 2k + 2$, $k = 0,1,2\ldots$.

Let $j_1, j_2, \ldots, j_n$ be a sequence satisfying Lemma 3.3.

Step 1. For each i we compute $y_{i-k,i}, y_{i-k+1,i}, \ldots, y_{i+k,i}$ by the basic construction computing the diagonal element y_{ii} by

$$\begin{array}{ll} A(a_i,x_i) + B(a_i,x_i) & \text{if } j_i = 1 \\ -C(a_i,x_i) + D(a_i,x_i) & \text{if } j_i = 2 \\ E(a_i,x_i) + F(a_i,x_i) & \text{if } j_i = 3 \end{array}$$

This requires $(3mn-2n)/2$ multiplications.

Step 2. Compute the pairs of elements $y_{k+2,1}$ and $y_{1,k+2}$, $y_{k+3,2}$ and $y_{2,k+3}, \ldots, y_{2k+2,k+1}$ and $y_{k+1,2k+2}$. Also, if $3k + 4 \leq n$, compute $y_{3k+4,2k+3}$ and $y_{2k+3,3k+4}$, $y_{3k+5,2k+4}$ and $y_{2k+4,3k+5}, \ldots,$ $y_{n,n-k-1}$ and $y_{n-k-1,n}$. This computation is done by the same method that was used for computing pairs of off-diagonal elements in the basic construction (i.e. using Lemma 3.2 if the corresponding diagonal elements are computed by distinct methods, and computing the adjacent elements otherwise) at a cost of three multiplications per pair.

Step 3. If $2k+2 \leq n \leq 3k+3$, let $\ell = 2k+3$ and $p' = n - (2k+2)$. If $3k+3 < n < 4k+4$, then $\ell = n - k$ and $p' = 4k + 4 - n$. Now all columns have m consecutive elements computed except for columns $\ell, \ell+1, \ldots, \ell+p'-1$ which have only $m - 1$ consecutive elements computed. If p' is odd compute $y_{\ell+p'+k,\ell+p'-1}$ as $A(a_{\ell+p'+k},x_{\ell+p'-1}) + B(a_{\ell+p'+k},x_{\ell+p'-1})$ and let $p = p' - 1$. If p' is even, let $p = p'$. We now must compute one more independent element in columns $\ell,\ell+1,\ldots,\ell+p-1$. We will compute the pairs of elements $y_{\ell+k+1+2i,\ell+2i}$ and $y_{\ell-k+2i,\ell+1+2i}$ for $0 \leq i \leq (p-2)/2$. Let $i_1 = \ell+k+1+2i$, $i_2 = \ell+2i$, $i_3 = \ell-k+2i$ and $i_4 = \ell+1+2i$. For $0 \leq i \leq (p-2)/2$ we compute $y_{i_1i_2}$ and $y_{i_3i_4}$ by computing the four elements

$$Z(i_2+i_3,i_2+i_3) = y_{i_2i_2} + y_{i_2i_3} + y_{i_3i_2} + y_{i_3i_3}$$

$$Z(i_2+i_3,i_1+i_4) = y_{i_2i_1} + y_{i_2i_4} + y_{i_3i_1} + y_{i_3i_4}$$

$$Z(i_1+i_4,i_2+i_3) = y_{i_1i_2} + y_{i_1i_3} + y_{i_4i_2} + y_{i_4i_3}$$

$$Z(i_1+i_4,i_1+i_4) = y_{i_1i_1} + y_{i_1i_4} + y_{i_4i_1} + y_{i_4i_4}$$

Note that for $0 \leq i \leq (p-2)/2$, $0 \leq i_3 \leq \ell$, $\ell+p-1 < i_1$ and $i_1-n < \ell$. Thus every element in columns i_1 and i_3 is a linear combination of already computed elements and since $y_{i_2i_4}$ and $y_{i_4i_2}$ were computed in step 1, we can obtain $y_{i_1i_2}$ from $Z(i_1+i_4,i_2+i_3)$ and $y_{i_3i_4}$

from $Z(i_2+i_3, i_1+i_4)$. (The case $k = 0$ is vacuously true.)

The elements $Z(i_2+i_3, i_2+i_3)$, $Z(i_2+i_3, i_1+i_4)$, $Z(i_1+i_4, i_2+i_3)$ and $Z(i_1+i_4, i_1+i_4)$ are computed using seven multiplications by Lemma 3.2. The element $Z(i_2+i_3, i_2+i_3)$ is computed using two multiplications according to method j ($j = 1,2,$ or 3) of Lemma 3.2 where j is selected so that $j \neq j_2$ and $j \neq j_3$ where j_2 and j_3 are the methods of Lemma 3.2 used to compute the diagonal elements $y_{i_2 i_2}$ and $y_{i_3 i_3}$, respectively. Similarly $Z(i_1+i_4, i_1+i_4)$ is computed by two multiplications by method j' where j' is selected so that $j' \neq j_1$ and $j' \neq j_4$ where j_1 and j_4 are the methods of Lemma 3.2 used to compute $y_{i_1 i_1}$ and $y_{i_4 i_4}$. By (3) of Lemma 3.3 $j \neq j'$.

The claim is made that the two multiplications used to compute $Z(i_2+i_3, i_2+i_3)$ are already present since they are used to compute $y_{i_2 i_3}$ and $y_{i_3 i_2}$ in step 1. Similarly, the two multiplications used to compute $Z(i_1+i_4, i_1+i_4)$ are already present. Thus only three additional multiplications rather than seven are needed to compute the Z's . This completes the construction.

Two multiplications were used for each of the n diagonal elements and three multiplications for each of $\lfloor (mn-n)/2 \rfloor$ pairs of elements. For m even and n odd two additional multiplications were required for the left over element. Thus a total of $\lceil (3mn+n)/2 \rceil$ multiplications were used. $\square$

CHAPTER 4
NONCOMMUTATIVE RINGS — LOWER BOUND

We now turn our attention to the determination of a lower bound on the complexity of matrix multiplication for noncommutative rings. It will be shown that any program which computes $M(2,2,n)$ for all noncommutative rings requires at least $\left\lceil \frac{7n}{2} \right\rceil$ multiplications. This, combined with the algorithm of the previous chapter, is sufficent to prove that the complexity of $M(2,2,n)$ is exactly $\left\lceil \frac{7n}{2} \right\rceil$. A proof will then be given that the algorithm is also optimal for $M(3,2,3)$.

Without loss of generality we can make some assumptions about the form of a program if it computes the matrix multiplication problem for all noncommutative rings. In particular, we can assume that each output is computed as a linear sum of p[illegible]oducts, where each product is the product of two line[illegible] sums of the input variables. (Here we take "sum" [illegible] mean both addition and substraction.) We can make [illegible] assumption because, since the program is valid f[illegible] noncommutative rings, it must be valid for the r[illegible]f integers under addition and multiplication. I[illegible]is case we know that an output of the program, if it is to equal M_{ij} which consists only of second order terms, must also consist only of second order terms. Furthermore, if cubic or higher order terms appear in intermediate steps in the computation, the only possibility preventing them from appearing in the final result is their cancellation by the addition or subtraction of identical terms. Now assume that a previously computed product appears in another product. Then a cubic term will appear in the product. Since no cubic term can appear in the final result, if we delete all terms other than the quadratic terms the result will not be changed and no additional multiplications are required.

Since the program must compute the matrix multiplication problem for algebras such as matrices under matrix addition and multiplication in which the multiplication operation is not commutative, we can assume an even more restricted general form. The general form of the optimal program computes each M_{ij} as a linear sum of products of the form $\alpha\beta$, where α is a linear sum of a_{ij}'s and β is a linear sum of x_{ij}'s . Because $x_{ij}a_{k\ell}$ is not in general equal to $a_{k\ell}x_{ij}$, any products not of the above form will produce extraneous terms not in the final result. Removing all a_{ij}'s from right-hand multiplicands and x_{ij}'s from left-hand multiplicands will transform any program for matrix multiplication for noncommutative rings into the above form without increasing the number of multiplications and will leave the output of the program unaltered.

Finally, for the remainder of the chapter, the assumption is made that the algebra under consideration is one in which each element is its own additive inverse. An example of such an algebra is the binary Boolean algebra. We can make this assumption because, since the program is valid for all rings, then it must be valid for this particular ring. For this type of algebra we can replace any subtractions in a program by additions. Therefore the general form of the optimal program is as described above, where "linear sum" can now be taken to mean additions only.

Two results are now presented which will be needed later. Let $\mathfrak{F}$ be a field and $\{a_1, \ldots, a_n\}$ a set of indeterminates. Let $\hat{\mathfrak{F}}$ be the field obtained by extending $\mathfrak{F}$ by rational expressions of the a_i's . A set of vectors $x_1, x_2, \ldots, x_p$ with elements from $\hat{\mathfrak{F}}$ are <u>non-dependent</u> if and only if $\sum_{i=1}^{p} a_i x_i = 0$ and each a_i in $\mathfrak{F}$ implies each $a_i = 0$. Nondependence should not be confused with linear independence where the a_i's are not restricted to $\mathcal{F}$ but can be arbitrary elements of $\hat{\mathcal{F}}$.

Lemma 4.1: Let $F = \{f_1,..,f_k,...,f_n\}$ be a set of expressions, where $f_1,...,f_k$ are independent and each can be expressed as a single product. If F can be computed with p multiplications, then there exists an algorithm for F with p multiplications in which k of the multiplications are $f_1,...,f_k$.

Proof: Let $m_1,...,m_p$ be the multiplications of some algorithm for F . Since $f_1,...,f_k$ are independent we can solve for k of the m_i's in terms of $f_1,...,f_k$ and the rest of the m_i's . □

Lemma 4.2: Any program for

$$\left.\begin{array}{l} a_{21}x_{2i} \\ a_{21}x_{1i} + a_{22}x_{2i} \end{array}\right\} 1 \leq i \leq n$$

requires $3n$ multiplications.

Proof: We can sssume by the above lemma that n of the multiplications are $a_{21}x_{2i}$, $1 \leq i \leq n$. If we remove these multiplications from the algorithm, it will compute $a_{21}x_{1i} + a_{22}x_{2i} + [a_{21}x_{21}] + \ldots + [a_{21}x_{2n}]$ $1 \leq i \leq n$, where $[\alpha]$ means that α may or may not be present.

In matrix form this can be represented as

$$\begin{bmatrix} a_{21} & 0 & a_{22} + [a_{21}] & [a_{21}] \\ \cdot & & \cdot & \\ \cdot & & \cdot & \\ 0 & a_{21} & [a_{21}] & a_{22}+[a_{21}] \end{bmatrix} \begin{bmatrix} x_{11} \\ x_{1n} \\ x_{21} \\ x_{2n} \end{bmatrix}$$

The first matrix clearly has $2n$ nondependent columns, therefore by Theorem 2.2 for this computation $2n$ multiplications are required. Since n multiplications were removed from the original algorithm, it must have had $3n$ multiplications $\square$

In what follows we will make use of the fact that after computing a set of expressions, we can with no additional multiplications compute all linear sums of the expressions.

Theorem 4.1: Let A and B be two sets of expressions where each expression in B is some linear combination of the expressions in A . If the computation of B requires p multiplications, then the computation of A requires p multiplications.

Proof: Suppose there is an algorithm for A containing fewer than p multiplications. Then this algorithm can be used to compute B with fewer than p multiplications simply by appending the necessary additions to compute the linear combinations.

We now develop the concepts necessary to take advantage of the high degree of symmetry in the matrix multiplication problem. We denote by $\mathcal{L}\{a_{ij}\}$ the set of all linear functionals of a_{ij}'s . A <u>transformation</u> T is a mapping $T : \{a_{ij}\} \cup \{x_{k\ell}\} \to \mathcal{L}[\{a_{ij}\} \cup \{x_{k\ell}\}]$. $A \times X$ is said to be <u>invariant</u> under a transformation T if $T(\mathcal{L}\{A \times X\}) = \mathcal{L}\{A \times X\}$. An example of such a transformation is

$$\begin{array}{lll} a_{1j} \to a_{1j} + a_{2j} & & 1 \le j \le 2 \\ a_{ij} \to a_{ij} & & 2 \le i \le n \ , \ 1 \le j \le 2 \\ x_{ij} \to x_{ij} & & 1 \le i \le 2 \ , \ 1 \le j \le n \end{array}$$

The set $\mathcal{T}$ of all transformations which leave $A \times X$ invariant forms a group. Thus, we can use $\mathcal{T}$ to define an equivalence relation on multiplications as follows: If $m_1 = \alpha \cdot \beta$ and $m_2 = \gamma \cdot \delta$ are two multiplications and there exists some T in $\mathcal{T}$ such that $T(\alpha) = \gamma$, then $m_1 \equiv m_2$. Since $\mathcal{T}$ is a group, $\equiv$ as defined above is, in fact, an equivalence relation. Under this equivalence relation, the multiplications used in the computation of $A \times X$ form n equivalence classes where n is the dimension of the largest square matrix conatined in A . Representative members of these classes are $a_{11}\beta, (a_{11}+a_{22})\beta, \ldots, (a_{11}+a_{22}+\ldots+a_{nn})\beta$, for any β in $\mathcal{L}\{x_{ij}\}$.

The next technique, similar to one used by Pan [2], is to reduce the number of variables by selecting a linear sum of variables $x_1+x_2+\ldots+x_n$ which appears as a multiplicand and setting the sum equal to zero. This can be done by solving the resulting linear equation for one of the variables, say x_1 , and substituting the solution $x_1 = x_2+\ldots+x_n$ for that variable whenever it occurs in the algorithm. The result is a new algorithm P' which has at least one less multiplication than the original algorithm P . If P computed $f_i(x_1,x_2,\ldots,x_n)$, $1 \leq i \leq m$, then P' computes $f'_i(x_2,\ldots,x_n) = f_i(x_2+\ldots x_n,x_2,\ldots,x_n)$, $1 \leq i \leq m$. We conclude that the optimal algorithm for the f's has a least one more multiplication than the optimal algorithm for the f' 's.

Combining this technique with the symmetry argument results in a powerful tool. For example, in computing $A \times X$ where A and X are both 2×2 matrices, we can assume by symmetry that either a_{11} or

$a_{11} + a_{22}$ appears as a multiplicand. Thus, by substituting first $a_{11} = 0$ and then $a_{11} = a_{22}$, we deduce that the minimal number of multiplications for the above computation is more than the minimal number for one of

$$\begin{array}{llll}
 & \text{a)} & a_{12}x_{21} & a_{12}x_{22} \\
 & & a_{21}x_{11} + a_{22}x_{21} & a_{21}x_{12} + a_{22}x_{22} \\
\text{or} & \text{b)} & a_{22}x_{11} + a_{12}x_{21} & a_{22}x_{12} + a_{12}x_{22} \\
 & & a_{21}x_{11} + a_{22}x_{21} & a_{21}x_{12} + a_{22}x_{22}
\end{array}$$

These techniques can be used to show that multiplying a 2×2 matrix A by a $2 \times n$ matrix X requires exactly $\left\lceil \frac{7n}{2} \right\rceil$ multiplications.

<u>Lemma 4.3</u>: If an algorithm for $A \times X$ had k multiplications of forms $a_{11}\alpha$, $(a_{12}+a_{21})\beta$ and $(a_{11}+a_{12}+a_{21})\gamma$, then the algorithm requires at least $3n + k$ multiplications.

<u>Proof</u>: Set $a_{11} = 0$ and $a_{12} = a_{21}$. This eliminates k multiplications and the algorithm now computes

$$\left.\begin{array}{l} a_{21}x_{2i} \\ a_{21}x_{1i} + a_{22}x_{2i} \end{array}\right\} \quad 1 \le i \le n$$

which requires $3n$ multiplications by Lemma 4.2.

<u>Corollary</u>: By applying transformations in $\mathcal{J}$ we also have similar theorems for

a) $(a_{11}+a_{21})\alpha$, $(a_{12}+a_{21}+a_{22})\beta$, $(a_{11}+a_{12}+a_{22})\gamma$

b) $(a_{11}+a_{12})\alpha$, $(a_{12}+a_{21}+a_{22})\beta$, $(a_{11}+a_{21}+a_{22})\gamma$

c) $(a_{11}+a_{12}+a_{21}+a_{22})\alpha$, $(a_{12}+a_{21})\beta$, $(a_{11}+a_{22})\gamma$

d) $a_{21}\alpha$, $(a_{11}+a_{22})\beta$, $(a_{11}+a_{21}+a_{22})\gamma$

e) $(a_{21}+a_{22})\alpha$, $(a_{11}+a_{12}+a_{22})\beta$, $(a_{11}+a_{12}+a_{21})\gamma$

f) $a_{12}\alpha$, $(a_{11}+a_{22})\beta$, $(a_{11}+a_{12}+a_{22})\gamma$

g) $(a_{12}+a_{22})\alpha$, $(a_{11}+a_{21}+a_{22})\beta$, $(a_{11}+a_{12}+a_{21})\gamma$

h) $a_{22}\alpha$, $(a_{12}+a_{21})\beta$, $(a_{12}+a_{21}+a_{22})\gamma$

<u>Lemma 4.4</u>: Any algorithm P for $A \times X$ which has k multiplications of type $a_{11}\alpha$, $a_{12}\beta$ and $(a_{11}+a_{12})\gamma$ has at least $3n + \frac{k}{2}$ multiplications.

<u>Proof</u>: Let L be the vector space of linear combinations of the k multiplications which are assumed to be present. We first prove the theorem for the case where the dimension of L is k .

Consider the set of expressions $S' = \{a_{11}x_{1i}+a_{12}x_{2i} \mid 1 \leq i \leq n\}$, each of which is computed by P . Let S be the vector space of linear combinations of expressions in S' . Clearly S has dimension

n . The subspace $S \cap L$ has dimension less than or equal to $\frac{k}{2}$, since any $\frac{\ell}{2}$ linearly independent expressions in $S \cap L$ can be represented as the product of a matrix with ℓ non-dependent columns and an arbitrary vector and thus requires ℓ multiplications by Theorem 2.2. Hence, the vector space spanned by $S \cup L$ has dimension of at least $n + \frac{k}{2}$. We can therefore solve for $n + \frac{k}{2}$ of the multiplications in the algorithm P in terms of the expressions in S' and the k multiplications. If we now set $a_{11} = a_{12} = 0$, then we lose $n + \frac{k}{2}$ multiplications from P .

The new algorithm computes $a_{21}x_{1i} + a_{22}x_{2i}$ $1 \leq i \leq n$ which requires $2n$ multiplications by Theorem 2.2. Hence P must have had at least $3n + \frac{k}{2}$ multiplications.

In the case where the dimension of M is k' , $k' < k$, then solve for $k - k'$ multiplications in terms of the remaining multiplications and replace them by their solution wherever they appear in the program. This results in an algorithm with at least $3n + \frac{k'}{2}$ multiplications as we have just proved. Thus P must have had at least $3n + \frac{k'}{2} + k - k' > 3n + \frac{k}{2}$

multiplications. $\square$

<u>Corollary</u>: By transformations we also have similar theorems for $a_{21}\alpha$, $a_{22}\beta$, $(a_{21}+a_{22})\gamma$ and $(a_{11}+a_{21})\alpha$, $(a_{12}+a_{22})\beta$, $(a_{11}+a_{12}+a_{21}+a_{22})\gamma$.

<u>Theorem 4.2</u>: The complexity of $M(2,2,n)$ for noncommutative rings is at least $\left\lceil \frac{7n}{2} \right\rceil$.

<u>Proof</u>: Divide the multiplications in P into two disjoint sets S_A and S_B , where the multiplications in S_A begin with a_{11} , a_{12} , a_{21} , a_{22} , $a_{11}+a_{12}$, $a_{11}+a_{21}$, $a_{12}+a_{22}$, $a_{21}+a_{22}$ or $a_{11}+a_{12}+a_{21}+a_{22}$ and the multiplications in S_B begin with $a_{11}+a_{22}$, $a_{12}+a_{21}$, $a_{11}+a_{12}+a_{21}$, $a_{11}+a_{12}+a_{22}$, $a_{11}+a_{21}+a_{22}$ or $a_{12}+a_{21}+a_{22}$. Note that each multiplication in S_A belongs to one group of Lemma 4.3 and one group of Lemma 4.4, while each multiplication in S_B belongs to three groups of Lemma 4.3.

Suppose there are m_A multiplications in S_A and m_B multiplications in S_B . Let there be a_1 multiplications in the first group of Lemma 4.4, a_2 multiplications in the second group, and a_3 in the third. Since each multiplication in S_A belongs to one of these groups, then

$$a_1 + a_2 + a_3 = m_A .$$

Therefore one of a_1 , a_2 or a_3 must be greater than or equal to $m_A/3$, i.e. there is some group of Lemma 4.4 with at least $m_A/3$ multiplications.

Similarly, let $b_1,\ldots,b_9$ represent the number of multiplications in each of the nine groups of Lemma 4.3. Since each multiplication in S_A belongs to one of these groups and each multiplication in S_B belongs to three of them, then

$$b_1 + \ldots + b_9 = m_A + 3m_B .$$

Hence there must be some group of Lemma 4.3 with at least $(m_A + 3m_B)/9$ multiplications.

Suppose P has at least m multiplications. Since $m = m_A + m_B$, then either $m_A \geq 6m/7$ or $m_B \geq m/7$. If $m_A \geq 6m/7$, then there must be some group of Lemma 4.4 with at least $m_A/3 \geq 2m/7$ multiplications, and hence by Lemma 4.4 $m \geq 3n + m/7$. If $m_B \geq m/7$, then there must be some group of Lemma 4.3 with at least $(m_A+3m_B)/9 = ((m-m_B)+3m_B)/9 \geq m/7$ multiplications and hence by Lemma 4.3 $m \geq 3n + \frac{m}{7}$. In either case, therefore,

$m \geq 3n + \frac{m}{7}$. Solving this inequality, we find that

$m \geq \frac{7n}{2}$. □

We now proceed with a series of lemmas which will provide the basis for proving that any algorithm for $M(3,2,3)$ requires at least 15 multiplications, and hence the algorithm of Chapter 3 is optimal for this case.

<u>Lemma 4.5</u>: Any algorithm for $M(3,2,3)$ with a multiplication equivalent (by a transformation in $\mathcal{J}$) to $a_{31}\beta$ has at least 15 multiplications.

<u>Proof</u>: Assume that the algorithm has only 14 multiplications. Assume without loss of generality that $a_{31}\beta$ is present. Set $a_{31} = 0$. Then

$$\left.\begin{array}{l} a_{11}x_{1j} + a_{12}x_{2j} \\ a_{21}x_{1j} + a_{22}x_{2j} \\ a_{32}x_{2j} \end{array}\right\} 1 \leq j \leq 3$$

must be computed in 13 multiplications. By Lemma 4.1 we can assume that the multiplications $a_{32}x_{2j}$, $1 \leq j \leq 3$, are present. Set $a_{32} = 0$ and we have an algorithm computing $a_{i1}x_{1j} + a_{i2}x_{2j}$, $1 \leq i \leq 2$, $1 \leq j \leq 3$, in 10 multiplications contradicting Theorem 4.2. □

Lemma 4.6: If an algorithm for $M(3,2,3)$ has two multiplications of type $(a_{11}+a_{22})\alpha$ and $(a_{12}+a_{21})\beta$, then the algorithm has at least 15 multiplications.

Proof: Set $a_{11} = a_{22}$ and $a_{12} = a_{21}$, losing two multiplications. Without loss of generality we can assume that $a_{32}\beta$ is present, because any multiplication involving a_{32} is equivalent to $a_{32}\beta$ by some transformation in $\mathcal{J}$. Set $a_{32} = 0$, losing one multiplication. By Lemma 4.1 setting $a_{31} = 0$ costs three multiplications. It can be shown that what remains requires 9 multiplications, and therefore $9+3+1+2 = 15$ multiplications were required originally. □

Corollary: By transformations we obtain similar theorems for the groups $a_{11}+a_{22}$ and $a_{11}+a_{21}+a_{22}$, and $a_{11}+a_{22}$ and $a_{11}+a_{12}+a_{22}$.

Lemma 4.7: If an algorithm for $M(3,2,3)$ has two multiplications of the type $(a_{11}+a_{22})\alpha$, $(a_{11}+a_{12}+a_{21})\beta$ and $(a_{12}+a_{21}+a_{22})\gamma$, then the algorithm has at least 15 multiplications.

Proof: Set $a_{11} = a_{22}$ and $a_{12} = a_{21} + a_{22}$, losing two multiplications. In the algorithm so obtained some multiplication must involve a_{22}; let it be $(a_{22}+\alpha)\beta$

where α is one of eight possible cases. For each case it can be shown that after setting $a_{22} = \alpha$, which costs one multiplication, the remainder requires 12 multiplications. Hence the original algorithm must have had at least $12+1+2 = 15$ multiplications. □

Lemma 4.8: If an algorithm for $M(3,2,3)$ has two multiplications of type $(a_{11}+a_{22})\alpha$, $(a_{11}+a_{12}+a_{22})\beta$, $(a_{12}+a_{21}+a_{22})\gamma$, $(a_{12}+a_{21})\delta$, $(a_{11}+a_{12}+a_{21})\varepsilon$ and $(a_{11}+a_{21}+a_{22})\zeta$, then the algorithm has at least 15 multiplications.

Proof: Without loss of generality assume the first of the two multiplications is $(a_{11}+a_{22})\alpha$. If the second multiplication is $(a_{11}+a_{22})\beta$, $(a_{11}+a_{12}+a_{21})\beta$ or $(a_{12}+a_{21}+a_{22})\beta$, then the algorithm has at least 15 multiplications by Lemma 4.7. If the second multiplication is $(a_{12}+a_{21})\beta$, $(a_{11}+a_{21}+a_{22})\beta$ or $(a_{11}+a_{12}+a_{22})\beta$, then the algorithm has at least 15 multiplications by Lemma 4.6 and its corollary. □

Corollary: The Lemma holds for seven groups of six multiplications each, which covers all 42 types of multiplications not equivalent to $a_{31}\beta$.

Lemma 4.9: If an algorithm for $M(3,2,3)$ has eight or more multiplications not equivalent to $a_{31}\beta$, then the algorithm has at least 15 multiplications.

Proof: At least one group of the corollary to Lemma 4.8 must have two or more multiplications, and the proof follows immediately from Lemma 4.8. □

Theorem 4.3: Any program computing $M(3,2,3)$ for noncommutative rings requires 15 multiplications.

Proof: The program requires at least 11 multiplications to compute the first two rows by Theorem 4.2. If one of these is equivalent to $a_{31}\beta$, then the theorem is true by Lemma 4.5. If none of the multiplications is equivalent to $a_{31}\beta$, then the theorem is true by Lemma 4.9. □

CHAPTER 5
COMMUTATIVE RINGS

Strassen's method and the algorithm of Chapter 3 provide improved matrix multiplication programs for structures which satisfy all the axioms of a noncommutative ring. We now show that even more efficient programs exist providing that the multiplication operation is commutative. Specifically, we are now dealing with structures which satisfy the following axioms:

Addition: associative, commutative, identity, inverse;

Multiplication: associative, distributive, commutative, identity.

<u>Theorem 5.1</u>: The complexity of the matrix multiplication problem $M(m,2,n)$ for commutative rings is no more than $mn + m + n$.

<u>Proof</u>: The theorem can be proved by describing a method due to Winograd [5] which computes the product $A \times X$, where A is an $m \times 2$ matrix and X is a $2 \times n$ matrix, using just $mn + m + n$ multiplications. The method proceeds as follows:

1) Compute the m products $a_{i1}a_{i2}$, $1 \leq i \leq m$.

2) Compute the n products $x_{1j}x_{2j}$, $1 \leq j \leq n$.

3) Compute the mn products $(a_{i1} + x_{2j})(a_{i2} + a_{1j})$, $1 \leq i \leq m$, $1 \leq j \leq n$.

4) If $A \times X = Y$, then compute the y_{ij}'s from the above products by

$$y_{ij} = (a_{i1}+x_{2j})(a_{i2}+x_{1j}) - a_{i1}a_{i2} - x_{1j}x_{2j} .$$

If can be readily be proved from the axioms of a commutative ring that the program does compute $M(m,2,n)$ for all commutative rings, and it is clear that $mn + m + n$ multiplications are used. $\square$

If we set $m = 2$ and compare this bound with the minimal bound of $\left\lceil \frac{7n}{2} \right\rceil$ for noncommutative rings, we find that Winograd's method requires fewer multiplications for all $n \geq 5$. We now ask ourselves how good is Winograd's method; in other words what is a lower bound on the complexity of matrix multiplication for commutative rings? A rough estimate can be obtained as follows:

<u>Lemma 5.1</u>: The complexity of $M(m,p,n)$ is at least $(m + n - 1)p$.

<u>Proof</u>: By an argument similar to the noncommutative case, one can assume without loss of generality that the optimal program for $M(m,p,n)$ computes expressions in the form

of a sum of products of sums of a_{ij}'s and x_{ij}'s . Briefly this is true because only second degree terms appear in $M(m,p,n)$.

Let P be the optimal program for $M(m,p,n)$. Since P computes $M_{11} = a_{11}x_{11}+\ldots+a_{1p}x_{p1}$, P must have some multiplication of the form $(a_{11}+\alpha)(x_{11}+\beta)$ in order to obtain the term $a_{11}x_{11}$. Setting $a_{11} = -\alpha$ we obtain a new program P' which has one multiplication fewer than P and which computes $M'_{11} = -\alpha x_{11}+a_{12}x_{21}+\ldots+a_{1p}x_{p1}$ instead of M_{11} . Regardless of what α is, P' must have a multiplication involving a_{12} in order to obtain the term $a_{12}x_{21}$ in M'_{11} . Upon removing this multiplication by substituting for a_{12} we obtain a new program P'' which has two multiplications fewer than P . We continue this for each a_{1k} , $1 \leq k \leq p$, obtaining a program $P^{(p)}$ which has p multiplications fewer than P . Note that since we have only substituted for the a_{1k}'s , $P^{(p)}$ still computes M_{ij} , $2 \leq i \leq m$, $1 \leq j \leq n$.

Similarly, by considering M_{21} we can remove p more multiplications and the resulting program still computes M_{ij} , $3 \leq i \leq m$, $1 \leq j \leq n$. By repeating this for $M_{31}, \ldots, M_{m-1,1}$, we obtain a program which has $(m-1)p$ multiplications fewer than P and which computes M_{mj}, $1 \leq j \leq n$. By Theorem 2.2 this requires at least np multiplications; therefore P must have had at least $(m + n - 1)p$ multiplications.

Lemma 5.2: The complexity of $M(m,p,n)$ is at least $(m + p - 1)n$.

Proof: Let P be the optimal program for M. Since the set $S = \{M_{ij} | 1 \leq i \leq m-1 , 1 \leq n \leq n\}$ contains $(m-1)n$ linearly independent expressions each of which is computed by P, we can solve for $(m-1)n$ of the multiplications in P in terms of the other multiplications and the expressions in S. Therefore, by setting $a_{ik} = 0$ and $1 \leq i \leq m-1$ and $1 \leq k \leq p$, which has the effect of setting each expression in S equal to zero, we obtain a new program P' with $(m-1)n$ fewer multiplications than P and P' computes $M_{m1}, \ldots, M_{mn}$. By Theorem 2.2 P' must have at least np multiplications, so P must have had at least $(m-1)n + np = (m + p - 1)n$ multiplications □

Corollary: The complexity of $M(m,p,n)$ for commutative rings is at least $(n + p - 1)m$.

By combining these results we obtain the following:

Theorem 5.2: The complexity of $M(m,p,n)$ for commutative rings is at least $\max\{(m+n-1)p,(m+p-1)n,(n+p-1)m\}$.

CHAPTER 6

SOME OTHER ALGEBRAIC STRUCTURES

In Chapter 3 an algorithm was described which computes the matrix multiplication problem for all rings using fewer multiplications than one would ordinarily expect. It will now be shown that this improvement is highly dependent upon the ring axioms. In particular, if we remove either the additive inverse axiom or the distributivity axiom, the complexity of the matrix multiplication problem $M(m,p,n)$ becomes mpn , or the same as for an algebra satisfying no axioms at all.

We first examine algebras satisfying the following axioms:

Addition: associative, commutative, identity;

Multiplication: associative, distributive, commutative, identity.

In other words we have all the axioms of a ring with unity except the existence of an additive inverse (minus). We wish to show that the complexity of the matrix multiplication problem $M(m,p,n)$ for this type of structure is mpn . To show this it suffices to exhibit some algebra $\mathfrak{a} = \langle S, \oplus , \odot \rangle$ which satisfies the above axioms and prove that the complexity of $M(m,p,n)$ for $\mathcal{a}$ is mpn . There are several algebras for which one can prove this, such as the natural numbers under <u>max</u> and $+$ or the set of all subsets of $\{a\}^*$ under union and concatenation, but the one which is presented here is the set of natural numbers under $+$ and $\cdot$, or $< \mathbb{N},+,\cdot>$.

In order to prove that the complexity of $M(m,p,n)$ is mpn for $< \mathbb{N},+,\cdot>$, we first assume that there is some program P which computes $M(m,p,n)$. We then define a normal form into which one can transform any expression computed by P . Next we prove that if some expression E computed by P is equal to M_{ij} , then the normal form of E must be precisely $a_{i1}x_{1j}+\ldots+a_{ip}x_{pj}$. We then show that if the normal form of E has this form then each product $a_{ik}x_{kj}$, $1 \leq k \leq p$, is present in E . We can then conclude that each product $a_{ik}x_{kj}$, $1 \leq i \leq m$, $1 \leq k \leq p$, $1 \leq j \leq n$, is present in P , and therefore that P uses at least mpn multiplications.

<u>Definition</u>: The <u>normal form</u> of some expression E , denoted by $NF(E)$, is obtained as follows:

1) remove all parentheses according to the associativity axioms and the distributivity axiom;

2) rearrange the factors in each term into lexicographical order (this can be done because $\cdot$ is commutative);

3) rearrange the terms into lexicographical order ($+$ is commutative);

4) combine c identical terms α into one term $c \cdot \alpha$, e.g. $a_{11}x_{11} + a_{11}x_{11}$ becomes $2a_{11}x_{11}$ (this can be done according to the multiplicative identity axiom and the distributivity axiom).

<u>Lemma 6.1</u>: If $E = M_{ij}$, then there are no terms in $NF(E)$ of order three or more.

<u>Proof</u>: Assume there is a term in $NF(E)$ of the form $c\alpha_1\alpha_2\ldots\alpha_r$, $r \geq 3$, each α_i in $\{a_{ij}\} \cup \{x_{ij}\}$. Set each a_{ij} and x_{ij} equal to $p + 1$. Then $M_{ij} = p(p+1)^2$, but $E \geq (p+1)^3$. Therefore $E \neq M_{ij}$, which is a contradiction. □

<u>Lemma 6.2</u>: If $E = M_{ij}$, then there are no zero or first order terms in $NF(E)$.

<u>Proof</u>: Assume there is some term in $NF(E)$ of the form $c\alpha$. Set $\alpha = 1$ and all other a_{ij}'s and x_{ij}'s equal to 0 . Then $E = c$ and $M_{ij} = 0$; hence $E \neq M_{ij}$, which is a contradiction. Similarly, if there is a zero

order term (constant) in NF(E) , then setting all a_{ij}'s and x_{ij}'s equal to zero leads to the same contradiction. $\square$

Lemma 6.3: If $E = M_{ij}$, then NF(E) must be $a_{i1}x_{1j}+\ldots+a_{ip}x_{pj}$.

Proof: By Lemmas 6.1 and 6.2 there must be only second order terms in NF(E) . Consider the term $c\alpha\beta$, α and β in $\{a_{ij}\} \cup \{x_{ij}\}$. If we set α and β equal to 1 , and all other a_{ij}'s and x_{ij}'s equal to 0 , then $E = c$ and

$$M_{ij} = \begin{cases} 1 & \text{if } \alpha = a_{ik} \text{ and } \beta = x_{kj} ,\ 1 \le k \le p \\ 0 & \text{otherwise.} \end{cases}$$

Therefore $c = 1$ for the terms $a_{i1}x_{1j},\ldots,a_{ip}x_{pj}$ and 0 for all others. $\square$

Lemma 6.4: If P is a program which computes M(m,p,n) for $<\mathbb{N},+,\cdot>$ then P uses at least mpn multiplications.

Proof: The output of P is a set of expressions $\{E_{ij}\}$ such that $E_{ij} = M_{ij}$. By Lemma 6.3 we know that $NF(E_{ij})$ is $a_{i1}x_{1j}+\ldots+a_{ip}x_{pj}$. We can assume that for each k there is some product in E_{ij} of the form $(a_{ik}+\alpha)(x_{kj}+\beta)\gamma$, where α , β and γ are expressions, or a similar product with terms rearranged. If such a product did not appear, then $a_{ik}x_{kj}$ would not appear in $NE(E_{ij})$. We can immediately conclude that α and β are both zero, because otherwise extraneous terms would appear in $NF(E_{ij})$. By similar reasoning γ must be simply a constant (in fact $\gamma = 1$). In other words, the product $a_{ik}x_{kj}$ must appear in E_{ij} , and hence it appears as a multiplication in P . Since the above argument is valid for each product $a_{ik}x_{kj}$, $1 \leq i \leq m$, $1 \leq k \leq p$, $1 \leq j \leq n$, it follows that there must be at least mpn multiplications in P . □

This result leads to the following theorem:

Theorem 6.1: Any program P which computes the matrix multiplication problem M(m,p,n) for all "rings without minus" uses at least mpn multiplications.

Proof: Since $\mathcal{A} = \langle N,+,\cdot\rangle$ satisfies all the axioms of a "ring without minus", P must compute M(m,p,n) for $\mathcal{A}$. Then by Lemma 6.4 P must have at least mpn multiplications. □

It is interesting to see how this same result can be proved for yet another example of a "ring without minus" — again the set of natural numbers $\mathbf{N}$ but this time under the operations <u>min</u> and + instead of $\cdot$ and + . We will use $a \perp b$ to denote the minimum of a and b , i.e. $a \perp b \triangleq \min(a,b)$. The algebra $\mathcal{A} = \langle \mathbf{N}, \perp, + \rangle$ is especially useful in the important case of determining the shortest path matrix of a graph given its connection matrix. Improved matrix multiplication algorithms for $\mathcal{A}$ would lead to better techniques for determining the shortest path matrix than are presently available, but is will now be shown that the complexity of $M(m,p,n)$ for $\mathcal{A}$ is mpn , and therefore the ordinary method of matrix multiplication cannot be improved.

Assume that P is an optimal program for $M(m,p,n)$, i.e. P minimizes the number of + operations required and P computes a set of expressions $\{E_{ij} | 1 \leq i \leq m \ , \ 1 \leq j \leq n\}$ such that for all i and j

$$E_{ij} = M_{ij} = (a_{i1}+x_{1j}) \perp \cdots \perp (a_{ip}+x_{pj}) \ .$$

Of course while E_{ij} must equal M_{ij} , it need not have identically the same form as M_{ij} . We can, however, conclude certain facts about the possible form that E_{ij} can have. We first convert E_{ij} to an equivalent expression $\overline{E}_{ij}$ which has the form $S_1 \perp \ldots \perp S_t$ where $S_1, \ldots, S_t$ are sums of the input variables. This conversion is accomplished by the use of the distributivity axiom

$$a + (b \perp c) = (a + b) \perp (a + c)$$

in conjunction with the associativity and commutativity axioms.

We now show that $\overline{E}_{ij}$ must have the following form (with possibly a rearrangement of the terms):

$$\overline{E}_{ij} \equiv (a_{i1}+x_{1j}) \perp \ldots \perp (a_{ip}+x_{pj}) \perp (a_{i1}+x_{1j}+[\ ]) \perp \ldots \perp (a_{ip}+x_{pj}+[\ ])$$

where inside the brackets [ ] can be any expression at all. This means that each of the terms in $\overline{E}_{ij}$ must contain the sum of one of the pairs of variables a_{ik} and x_{kj}, and each of the terms $(a_{ik}+x_{kj})$ must be present in $\overline{E}_{ij}$. Assume that some term $(\alpha + \ldots + \gamma)$ which does not contain any pair a_{ik} and x_{kj} is present in $\overline{E}_{ij}$. Then setting $\alpha = \ldots = \gamma = 0$ and setting all the other variables equal to 1 leads to the contradictory conclusion that $E_{ij} = 0$ and $M_{ij} \geq 1$. Now assume that some term $(a_{ij}+x_{kj})$ does not appear in $\overline{E}_{ij}$. Setting $a_{ik} = x_{kj} = 0$ and all other variables equal to 1 leads to the conclusion that $M_{ij} = 0$ while $E_{ij} \geq 1$, which is also a contradiction. Hence the above form of $\overline{E}_{ij}$ is the only one possible.

Let us now examine how the terms $(a_{ik}+x_{kj})$ in $\overline{E}_{ij}$ could have been derived from E_{ij} by applications of the distributivity axiom. Any term which can be combined with this one must contain either a_{ik} or x_{kj} to provide the common factor, and the result after reducing them to a single term must be either $a_{ik} + (x_{kj} \perp [\ ])$ or $(a_{ik} \perp [\ ]) + x_{kj}$, where [ ] again represents any arbitrary expression. No matter how many times this reduction process is repeated, the resulting term must be of the form $(a_{ik} \perp [\ ]) + (x_{kj} \perp [\ ])$. We can therefore conclude that E_{ij} must have the following form:

$$E_{ij} \equiv \{(a_{i1} \perp [\ ])+(x_{1j} \perp [\ ])\} \perp \ldots \perp \{(a_{ip} \perp [\ ]) + (x_{pj} \perp [\ ])\} \perp [\ ] .$$

We now know that the optimal program P must contain each of the mpn additions

$$A_{ijk} = (a_{ik} \perp [\ ]) + (x_{kj} \perp [\ ]) .$$

There is still the possibility, however, that these additions are not all distinct. Perhaps for example, $A_{ijk} \equiv A_{uvw}$. For this to happen A_{ijk} must have a form like $(a_{ik} \perp \alpha \perp [\ ]) + (x_{kj} \perp [\ ])$, where α is a variable other than a_{ik} or x_{kj} . Let $a_{ik} = x_{kj} = 1$, let $\alpha = 0$, and let the rest of the variables equal 2 . We must then conclude that $M_{ij} = 2$ and $E_{ij} = 1$, which is a contradiction. Therefore P does contain mpn distinct additions.

We now turn our attention to algebras which satisfy all the axioms of a noncommutative ring with the exception that multiplication no longer distributes over addition. Specifically, we are dealing with algebras which satisfy the following axioms:

Addition: associative, commutative, identity, inverse;

Multiplication: associative.

An algebra satisfying these axioms will be referred to informally as a <u>nondistributive ring</u>. It will be shown that the complexity of M(m,p,n) for this algebraic structure is mpn .

We will actually prove this for a special type of nondistributive ring, which will be called a modified nondistributive ring, and the main result follows by

Theorem 2.1. A modified nondistributive ring satisfies all the above axioms with the added restriction that each element is its own additive inverse.

Theorem 6.2: The complexity of $M(m,p,n)$ for modified nondistributive rings is at least mpn (and therefore equal to mpn).

Proof: Consider the set E of all well-formed expressions over the symbols 0 and x , with operations $+$ and $\cdot$ and parentheses "(" and ")" . An equivalence relation can be defined over E as follows: Let e_1 and e_2 be two expressions in E . Then $e_1 \equiv e_2$ iff it can be proved from the modified nondistributive ring axioms that $e_1 = e_2$. It is easy to show that $\equiv$ is an equivalence relation (i.e. reflexive, symmetric and transitive). We denote by $\overline{E}$ the set of equivalence classes of E under the equivalence relation $\equiv$, and $\overline{E}(e)$ will be used to denote the equivalence class of expressions in E which are equivalent to e .

We can now describe the particular algebra which will be considered. The set of elements of the algebra is the set $\overline{E}$ of equivalence classes, while the addition and multiplication operations $\oplus$ and $\odot$ are defined as follows:

For any two elements $\overline{E}(e_1)$ and $\overline{E}(e_2)$,

$$\overline{E}(e_1) \oplus \overline{E}(e_2) = \overline{E}(e_1 + e_2)$$

and

$$\overline{E}(e_1) \odot \overline{E}(e_2) = \overline{E}(e_1 \cdot e_2) .$$

It is clear that the algebra $\mathcal{a} = <\overline{E}, \oplus , \odot >$ satisfies all the modified nondistributive ring axioms. Furthermore, as a result of the way in which $\overline{E}$ was defined, two expressions over $\oplus$ and $\odot$ are equal only if they can be proved to be equal by the modified nondistributive ring axioms.

Now suppose a program P computes $M(m,p,n)$ for $\mathcal{a}$. Then for each i and j , $1 \leq i \leq m$, $1 \leq j \leq n$, P must compute an expression F_{ij} which is equal to M_{ij} , where

$$M_{ij} = a_{i1} \odot x_{1j} \oplus \ldots \oplus a_{ip} \odot x_{pj} .$$

M_{ij} is equal to F_{ij} only if it can be proved to be equal from the axioms of a modified nondistributive ring. Let us apply, therefore, all possible axioms to M_{ij} , thereby obtaining all expressions which are equal to M_{ij} . The commutativity of addition allows rearranging the terms of M_{ij}, and the associativity of addition and multiplication merely permit an arbitrary order of evaluation of additions or multiplications which are not otherwise parenthesized. The additive identity and modified additive inverse axioms allow us to add expressions which always cancel out, but we can assume without loss of generality that such wasted terms do not appear in P. We are left, then, with the conclusion that F_{ij} contains each of the products $a_{i1} \odot x_{1j}, \ldots, a_{ip} \odot x_{pj}$, and therefore each of these products must be present in P. Since this is true for each i and j, there must be at least mpn products in P.

CHAPTER 7
CONCLUSION

The major results of the preceeding chapters can be summarized as follows:

1. A program is described for computing the product of an $m \times 2$ matrix and a $2 \times n$ matrix whenever the addition and multiplication operations satisfy all the axioms of a noncommutative ring, and the program uses no more than $\left\lceil \frac{3mn + \max(m,n)}{2} \right\rceil$ multiplications.

2. The computation of the product of a 2×2 matrix and a $2 \times n$ matrix for noncommutative rings requires at least $\left\lceil \frac{7n}{2} \right\rceil$ multiplications; therefore the algorithm referred to above minimizes the number of multiplications for at least these particular cases. It was also shown that 15 multiplications are required for the 3×2 by 2×3 case, and hence the algorithm is optimal in this instance as well.

3. The product of an $m \times 2$ matrix and a $2 \times n$ matrix can be computed for commutative rings with no more than $mn + m + n$ multiplications. In some cases (e.g. 2×2 by $2 \times n$, $n \geq 5$) this is less than can possibly be achieved for noncommutative rings, according

to the result above.

4. The product of an $m \times p$ matrix and a $p \times n$ matrix for commutative rings requires at least

$$\max\{(m+n-1)p, (m+p-1)n, (n+p-1)m\}$$

multiplications.

5. The product of an $m \times p$ matrix and a $p \times n$ matrix for "rings without minus" (i.e. all the ring axioms except additive inverse) requires exactly mpn multiplications.

6. The product of an $m \times p$ matrix and a $p \times n$ matrix for "nondistributive rings" (i.e. all the ring axioms except distributivity) requires exactly mpn multiplications.

In view of these results, an interesting complexity hierarchy is formed. At the top of the hierarchy is the class of all "free algebras", i.e. all algebras for which the addition and multiplication operations do not satisfy any axioms at all. Let this class be denoted by C_{free}. The complexity of this class is the maximum possible — mpn for the $m \times p$ by $p \times n$ case — and all other classes must either have the same complexity or lie below this class.

Let the class of all commutative and noncommutative rings be denoted by $C_{noncomm}$ and the class of all commutative rings by C_{comm}. Let the relation $C_a < C_b$ mean that the complexity of C_a is strictly less than C_b, and the relation $C_a = C_b$ mean that the complexities are identical. By Theorem 2.1 we know that

$$C_{comm} \leq C_{noncomm} \leq C_{free} .$$

Results 1, 2 and 3 above allow us to rewrite this with strict inequalities:

$$C_{comm} < C_{noncomm} < C_{free} .$$

Results 5 and 6 emphasize the dependence of Strassen's method on the ability to cancel unwanted terms in the computation. Strassen's method, and its generalization described in Chapter 3, are valid whenever all the axioms of a noncommutative ring are satisfied. If either the additive inverse axiom or the distributivity axiom is not satisfied, however, then there is no program which computes the $m \times p$ by $p \times n$ case with fewer than mpn multiplications for <u>all</u> such algebras.

It is possible, of course, that a <u>particular</u> algebra may not satisfy these axioms and yet have improved programs. Consider the set of integers I under addition and subtraction. For this algebra the "multiplication" operation (i.e. subtraction) is neither associative, distributive nor commutative. Yet we can compute the product of an $m \times p$ matrix and a $p \times n$ matrix for $<I,+,->$ with only mn "multiplications" (i.e. subtractions) because we can take advantage of the property that

$$(a - b) + (c - d) = (a + c) - (b + c) .$$

It is because of cases such as this that we were forced to define the complexity of a class of algebras to be the maximum complexity of all the algebras in the class.

One of the more interesting questions left unanswered by this thesis is whether any improvement can be obtained for matrix multiplication programs by allowing divisions as well as multiplications. In other words, what is the complexity of matrix multiplication for fields, where the complexity measure would not be the number of multiplications or divisions? This is probably a very difficult question to answer, but even limited results of this nature would contribute greatly to a more complete understanding of matrix multiplication.

A discouraging observation which must be made concerning this thesis research is the amount of tedious, combinatoric manipulation that was required to obtain such limited results — due mainly to the lack of powerful tools for proving lower bounds on the complexity of arithmetic functions. This is in contrast to problems such as sorting and searching which can be computed using only conditional transfer instructions. These functions can be analyzed thoroughly and elegantly using the tools of information theory. Especially useful is the concept of "entropy" as a measure of information, and a similar concept providing a measure of "computational effort" would probably represent the breakthrough which is required for the realistic analysis of arithmetic functions.

THE EFFECT OF ALGEBRAIC STRUCTURE ON THE COMPUTATIONAL COMPLEXITY OF MATRIX MULTIPLICATION

Leslie Robert Kerr, Ph.D.

Cornell University, 1970

Matrix multiplication is defined for two arbitrary operations $\oplus$ and $\odot$ as follows: The product of an $m \times p$ matrix A and a $p \times n$ matrix X is an $m \times n$ matrix Y such that

$$y_{ij} = (a_{i1} \odot x_{1j}) \oplus \dots \oplus (a_{ip} \odot x_{pj})$$

for $1 \leq i \leq m$, $1 \leq j \leq n$. The computational complexity of matrix multiplication is investigated with respect to the number of multiplications ($\odot$ operations) required in programs which are allowed additions, subtractions and multiplications, but no conditional transfer instructions.

It is shown that the complexity of matrix multiplication is highly dependent upon the algebraic structure of the two operations $\oplus$ and $\odot$. If $\oplus$ and $\odot$ satisfy all the axioms of a ring (not necessarily commutative) then a method due to Strassen can be used to multiply

$n \times n$ matrices using less than n^3 multiplications. For certain restricted cases, a lower bound is obtained on the number of multiplications required in any program (such as Strassen's method) which computes the product of two matrices for all rings. An extension of Strassen's method is described which meets this lower bound, and hence minimizes the number of multiplications used for at least these particular cases.

If $\oplus$ and $\odot$ satisfy all the axioms of a commutative ring, then a matrix multiplication algorithm due to Winograd can be used which requires fewer multiplications than the lower bound obtained for general rings. Hence the complexity of matrix multiplication, with respect to the number of multiplications required, is strictly less for commutative rings than for rings in general.

Finally, it is shown that if either the additive inverse axiom or the distributivity axiom is removed from the ring axioms, then in general the computation of the product of an $m \times p$ matrix and a $p \times n$ matrix requires mpn multiplications. For these cases, therefore, the ordinary method of computing matrix products cannot be improved.

BIBLIOGRAPHY

[1] Strassen, V. "Gaussian Elimination is not Optimal," unpublished manuscript, Angewandte Mathematik der Universitat Zurich, December 1968.

[2] Pan, V. Ya. "Methods of Computing Values of Polynomials," Russian Math. Surveys 21, pp. 105-136 (1966)

[3] Winograd, S. "On the Number of Multiplications Required to Compute Certain Functions," Proc. National Academy of Sciences 58:5, pp. 1840-1842 (1967).

[4] Blum, M. "A Machine Independent Theory of the Complexity of Recursive Functions," J. Assoc. Comp. Mach. 14, pp. 322-336 (1967).

[5] Winograd, S. "A New Algorithm for Inner Product," IEEE Trans. on Computers 17:7, pp. 693-694 (1968).

[6] Hopcroft, J.E. and Kerr, L.R. "Some Techniques for Proving Certain Simple Programs Optimal," Proc. Tenth Ann. Symp. on Switching and Automata Theory, pp. 36-45 (1969).

[7] Hopcroft, J.E. and Kerr, L.R. "On Minimizing the Number of Multiplications Necessary for Matrix Multiplication," Technical Report 69-44, Dept. of Computer Science, Cornell University (1969).