INDUCTIVE INFERENCE ON COMPUTER GENERATED PATTERNS

A thesis submitted to the Graduate School of the University of Wisconsin in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

by

Sister Mary Kenneth Keller BVM

Degree to be awarded

January 19—

June 19<u>65</u>

August 19—

To Professors: Hammer

Koenig

London

This thesis having been approved in respect to form and mechanical execution is referred to you for judgment upon its substantial merit.

Robert Alberty
Dean

Approved as satisfying in substance the doctoral thesis requirement of the University of Wisconsin.

Preston C. Hammer
Major Professor

Eldo C. Koenig

Ralph L. London.

Date of Examination, May 21 1965

INDUCTIVE INFERENCE ON COMPUTER GENERATED PATTERNS

by

SISTER MARY KENNETH KELLER B V M

A thesis submitted in partial fulfillment of the

requirements for the degree of

DOCTOR OF PHILOSOPHY

(Computer Sciences)

at the

UNIVERSITY OF WISCONSIN

1965

ACKNOWLEDGEMENTS

I wish to express my appreciation to Professor Preston C. Hammer for his suggestions which led to the selection of this area of research, his invaluable guidance in bringing it to completion, and especially for his patient assistance in my work over a period of years.

My gratitude is also extended to the Graduate Research Committee of the University of Wisconsin for the assistantship which supported my thesis research and made the computing facilities of the university available to me, and to the National Science Foundation for its support of my education.

Dr. J. W. Hanson, director of the University of North Carolina Computing Center, was kind enough to correspond with me regarding analytic differentiation by computer, and his contribution is gratefully acknowledged.

TABLE OF CONTENTS

INTRODUCTION

The basic problem of this thesis is the exploration of an approach to the mechanization of inductive inference. An inductive process is understood to mean, in this paper, the examination of specific cases with a view to making generalizations that may apply to a given set of which the known instances are members. These generalizations, if they can be formulated, are the result of a search for a pattern of change.

The program begins by generating a sequence of expressions. The definition of an expression is given in Chapter One, but for discussion purposes it may be thought of as a string of symbols connected by binary operators. The meaningful expressions are assumed, here, to be differentiable functions, and the successive differentiation carried out on an input expression generates expressions whose pattern of change is to be explored. The pattern program, however, was constructed to examine any set of expressions regardless of the rules under which they were generated. The search depends, for initiation, only on a specific representation of the expressions. This method of representation is a common method of syntactic analysis of input strings of symbols, sometimes referred to as Ershov's algorithm.[1]

Because the method of representation is a key choice with respect to the pattern search, the first chapter is devoted to a detailed discussion of a particular implementation. The next section deals with analytic differentiation based on the syntactic analysis described. The program to accomplish this is based on those of Hanson [2], B. W. Arden [3], and others. Those familiar with their methods may wish to turn to the third section where the discussion of the pattern search begins.

The main program accepts the first k derivatives of an input expression which are represented as a set of matrices, and by examining the sequences of operators and operands attempts to construct a description of their pattern of change, if one can be found. This description is stored in two matrices, one of which holds the information in the input matrices in a three dimensional array, and the other, the pattern of change descriptor, is formulated from the first. An auxiliary matrix, referred to as the "decision matrix" keeps track of the success of the search at each point in the array. A successful search, one in which a complete pattern is found, permits the construction of any n^{th} derivative, where $n > k$, of the original expression directly from the k^{th} derivative, by inference, and this last construction is also implemented by a machine program. Proof that this is the n^{th} derivative is not given in this paper, but it may be established by mathematical induction.

Input strings are written in Fortran with the usual interpretation. However, this restriction is not essential to what follows. Experimentation with format-free input subroutines shows that it is feasible to analyse data in ordinary algebraic notation, except, of course, for non-linear superscripts or subscripts. In Chapter One, there is a subroutine established to facilitate the handling of symbols in arrays, and to provide an interpretation.

The programming was carried out in CDC Fortran 63. Since the operations require manipulation of symbols, and what is, in effect, list-processing, languages such as IPL-V and LISP were considered because of their special capabilities in these areas. The decision made with regard to a programming language was based on its availability at the computing center.

CHAPTER ONE

Pattern Generation through the Analysis of Input Strings

An approach to the problem of mechanizing inductive inference can be made by developing subroutines for pattern detection, and change prediction, that parallel the activity of a mathematician who seeks an algorithmic definition of the n^{th} sentence of a relation which is already recursively defined. It need not be the case that the n^{th} sentence is unknown, in the sense that there is no path to its expression, but only that the labor involved is prohibitive. For example, by observing a changing pattern generated by recursion, mathematicians have constructed formulas such as the binomial expansion. It is safe to assume that the mental activity of the discoverers was initially that of inductive inference, and then a deductive proof was sought to support the statement.

There has been considerable interest in using the computer in theorem proving based on deductive reasoning.

For a survey of such algorithms, we refer to a paper by D. H. Potts.[4] To this we add the work of J. A. Robinson on machine-oriented logic.[5] It is noteworthy in the case of the investigations mentioned, that the theorem production is less interesting from the standpoint of originality than the special procedures developed for the machine. At first there is an attempt to imitate the human problem solver, and then to exploit some feature supplied by the use of a machine. For example, in his introduction, Robinson indicates this change of attitude and method:

> ". . . Traditionally a single step in a deduction has been required, for pragmatic and psychological reasons, to be simple enough, broadly speaking, to be apprehended as correct by a human being in a single gestalt. . . The 'single human gestalt' restriction is no longer very appropriate when the principles of inference are to be applied by a modern computing machine. More powerful principles, requiring perhaps a much greater amount of combinatorial data-processing for a single application, become a possibility. . ."

In the same fashion it is worthwhile to study the inductive processes of the human thinker, supposing at the same time that modification of some significance can be introduced through mechanization.

Accordingly, we wish to explore the use of the computer in inductive inference in two ways: (1) to mechanise repetitive processes in order to reduce burdensome symbol manipulation, and, at the same time, to increase accuracy; (2) to develop heuristic means of arriving at hypotheses more rapidly. To satisfy the first of these goals, it is necessary to develop a vocabulary to handle initializing expressions and their subsequent manipulations. What is done here, is to organize a vocabulary in such a way that it is useful directly in pattern detection. Moreover, in setting up this routine we also establish a means of pattern generation, as will be shown later.

Definition of an Expression

Expressions which serve as generators of recursively defined functions will be called data vectors to indicate that they represent input. The data vector is a string symbols defined as follows:

Using Backus Normal Form,[6] we define:

< VAR > ::= P|Q|R|S|T|U|V|W|X|Y|Z

< CON > ::= A|B|C|D|E|F|G|H|1|2|3|4|5|6|7|8|9|0

< FUN > ::= SIN|COS|TAN|COT|SEC|CSC|ASIN|ACOS|ATAN|
ACOT|ASEC|ACSC|SINH|COSH|TANH|COTH|SECH|
CSCH|ASINH|ACOSH|ATANH|ACOTH|ASECH|ACSCH|
LOG|EXP

< OP > ::= $ | (|) | - | + | * | / | NEG | ** | ·

Expressions are defined to be:

< DATA > ::= < CON > | < VAR > | < FUN > · < DATA > |
< DATA > + < DATA > | < DATA > - < DATA > |
< DATA > * < DATA > | < DATA > / < DATA > |
< DATA > ** < DATA > | NEG < DATA > | (< DATA >)

Expansion of this basic vocabulary will be noted in the sections appropriate to its need. All symbols defined will be referred to as elements of a language L.

The expected interpretation of these symbols is:

< VAR > :: = a variable

< CON > :: = a constant

< FUN > :: = a function name

< OP > :: = an operator, where $ is the delimiter of an expression, and * indicates multiplication, **, exponentiation, NEG represents unary negative, and "." is a pseudo-operator, introduced, following the suggestion of Hanson in his paper on analytic differentiation [2], to relate a function name to its argument.

< DATA > :: = an expression in L.

Syntactic Analysis of the Data Vector

Since we are interested in pattern detection, the method and form of storage of input is very relevant to the subsequent program. It turns out to be, in fact, a key selection. Ershov's algorithm [1] for syntactic analysis, adapted to a Fortran program, possesses characteristics which make it especially suitable for pattern generation. This adaptation is described as follows.

The data vector is analysed from right to left, and the elements are placed in what is equivalent to a push-down store, from which they are transferred according to operator precedence, to a matrix whose rows consist of an ordered pair of operands and their respective binary operator: $\theta(\alpha,\beta)$.

Let θ represent an operator in L, and $f(\theta)$ be its precedence level. Then we assign the range, $f(\theta)$, as follows:

f($) = 0.
f()) = 1.
f(() = 1.
f(-) = 2.
f(+) = 2.
f(*) = 3.
f(/) = 3.
f(NEG) = 4.
f(**) = 5.
f(·) = 6.

These values determine the respective precedence levels of the operators.

As the data vector is scanned from right to left, elements of L are placed in a push-down store in the order in which they are encountered, except for the symbols "\$", "(", ")". A precedence test is made for each operator, and a precedence list is created for the range of the operators in the same manner as for the symbol list. Then the following re-write rule holds:

> Let α, β be elements of the sets VAR, CON, or FUN in L, and θ an element of the set OP. Then $\theta(\alpha,\beta)$ is a binary operation in L.
>
> > $\theta_{i-1}(\alpha,\beta) \rightarrow R_j$ iff $f(\theta_i) < f(\theta_{i-1})$, and $\theta_i \neq$ ")", where j indicates the j^{th} re-write in the scan.

$\theta_{i-1}(\alpha,\beta)$ becomes the j^{th} row of the matrix M which is a parenthesis-free representation of the input expression with operator precedence preserved by means of the row order. The R_j's replace three symbols, i.e. a binary relation in L, in the push-down store, after which they are treated as regular symbols of L.

Example: (1)

Let the input vector be:

$(bx^2 - (\tan x)^2)/(ax^3 + \sin(cx))$.

Its representation in L is:

$(B*X**2 - (TAN · X)**2)/(A*X**3 + SIN · (C*X))$.

The syntactic scan produces a matrix which satisfies the re-write rule:

*(C,X)	→	R1
·(SIN,R1)	→	R2
**(X,3)	→	R3
*(A,R3)	→	R4
+(R4,R2)	→	R5
·(TAN,X)	→	R6
**(R6,2)	→	R7
**(X,2)	→	R8
*(B,R8)	→	R9
-(R9,R7)	→	R10
/(R10,R5)	→	R11

The symbols R1, R2, etc. are stored as part of the replacement vocabulary. The actual computer generated print-out looks like this:

C	*	X
SIN	·	R1
X	**	3
A	*	R3
R4	+	R2
TAN	·	X
R6	**	2
X	**	2
B	*	R8
R9	-	R7
R10	/	R5

It is clear that the re-write rule has been obeyed, and that a pattern has been generated.

<u>Diagram of the above scan:</u>

$$\underbrace{\$(\underbrace{\underbrace{B * \underbrace{X ** 2}_{R8}}_{R9} - \underbrace{(\underbrace{TAN \cdot X}_{R6}) ** 2}_{R7}}_{R10}))\;/\;(\underbrace{\underbrace{A * \underbrace{X ** 3}_{R3}}_{R4} + \underbrace{SIN \cdot \underbrace{(C*X)}_{R1}}_{R2}}_{R5})\$}_{R11}$$

The program which performs this analysis will be referred to as RAC (Read Analysis of Context). It is the first of the subroutines written with a view to pattern generation. Before continuing the discussion of the program itself, we should notice briefly how it might be considered a pattern generator. Suppose the data vectors produced are the successive binomial expansions of $(a + b)^n$. That is, the vectors: $a^2 + 2ab + b^2$, $a^3 + 3a^2b + 3ab^2 + b^3$, and so on. The subroutine RAC represents them as the following matrices:

(1)

B	**	2
A	*	2
R2	*	B
A	**	2
R4	+	R3
R5	+	R1

(2)

B	**	3
B	**	2
3	*	A
R3	*	R2
A	**	2
R5	*	B
R6	*	B
A	**	3
R8	+	R7
R9	+	R4
R10	+	R1

(3)

B	**	4
B	**	3
4	*	A
R3	*	R2
B	**	2
A	**	2
6	*	R6
R7	*	R5
A	*	3
R9	*	B
R10	*	4
A	**	4
R12	+	R11
R13	+	R8
R14	+	R4
R15	+	R1

In Chapter Three pattern detection is dealt with more fully, but the above example is inserted here to indicate the motivation for the selection of this method of storage. It is visually evident that there appears to be at least an operator pattern of change in the first three productions. Moreover, its detection could be mechanised very simply. Suppose that we use an alphabet consisting of operators -, +, *, /, NEG, **, and · , and ask that the pattern be expressed as a double-rowed list showing order and frequency of symbols from bottom to top. Then an induction is to be made on the order and frequency for the next row. If the induction holds, it is tested on the next row, and so on, until there is sufficient credibility to warrant a hypothesis.

Alphabet:		+	**	*	**	*	**	*	**	(*	**)
Frequency:	(1)	2	1	2	1						
	(2)	3	1	2	1	2	2				
	(3)	4	1	2	1	2	2	2	2		

The first hypothesis generated by the program is that the operator pattern of the next expansion is:

(4)	5	1	2	1	2	2	2	2	(2 | 2)

which, in fact, is correct. The final operator pattern to be induced is for $(a + b)^n$:

Alphabet:		+	**	*	**	*	**	*	**	(*	**)
Frequency:	(n - 1)	n	1	2	1	2	2	2	2 .	..2(n-2)	

repetitions.

Now if similar procedures are applied to the operand lists, the n^{th} matrix store can be constructed, and we have an algorithm for the n^{th} expansion in a new form, without proof, but with a weighted confirmability. It is also possible to search for a counterexample whose existence would destroy the hypothesis, regardless of the weight of confirmability built up.

The advantages, already evident, of this approach are that the operations are entirely mechanical, the n^{th} matrix can be produced by the computer, it can be output in fully parenthesized form, and finally, numerically evaluated for any parameters, and any N within machine capacity. In the example shown, this would be of little value since a well-known algorithm exists, but where this is not the case, it is an approach to mechanising hypothesis formation by inductive reasoning.

CHAPTER TWO

Analytic Differentiation as a Pattern Generator

The particular advantages of Ershov's algorithm[1] can be exploited in a useful application. A computer program for analytic differentiation may employ this algorithm effectively as has been demonstrated by B. W. Arden[3] in his text on digital computing, and in the previously mentioned work of Hanson[2]. The capacity of RAC to generate patterns is not limited to some particular type of subsequent re-write procedure such as the one which follows. Any other type of symbol manipulation which would result in another operator hierarchy-preserving matrix of the same form would serve as well.

It is possible to extend the language L by introducing a second set of re-write rules which will replace the input matrix with a derivative matrix. We remark that our chief concern is with pattern generation, and detection techniques, but there is an incentive for this specific application domain in that the algorithmic definition, in non-recursive form, of the n^{th} derivative of functions, and functions of functions, is not, in general, a fully solved problem. Even in cases where formulas for the n^{th} derivative exist there is not always a simple, operational statement available. Moreover, there is a very wide application for the n^{th} derivative in numerical analysis. It is hopeful that extensive experimentation with pattern detection on derivative matrices may be, at least, productive in labor reduction in actual calculations, and further, lead to the formation of new algorithms.

Program for Analytic Differentiation

The second program, preliminary to the induction experiments, resulted in the development of a subroutine which will be referred to as PAD (Program for Analytic Differentiation).

Analytic differentiation is accomplished by adding a second set of re-write rules to L. These re-write rules are based on the definitions of the first derivatives of the functions listed below, and on the grammar of L.

Let α, β be expressions of L, and $d\alpha$, $d\beta$ be their derivatives, then: for α,β = VAR, $d\alpha, d\beta = 1$, α,β = CON, $d\alpha, d\beta = 0$,

d(α + β) = dα + dβ.

d(α * β) = dα * β + α * dβ.

d(α / β) = (dα * β - α * dβ)/(β**2).

d(α ** β) = α ** β * (α*dβ/β + dα * LOG ·(α)).

d(SIN · (α)) = COS · (α) * dα.

d(COS · (α)) = -SIN · (α)* dα.

d(TAN · (α)) = SEC · (α) **2 * dα.

d(COT · (α)) = -(CSC · (α)) ** 2 * dα.

d(SEC · (α)) = TAN · (α) * SEC · (α) * dα.

d(CSC · (α)) = -CSC · (α) * SEC · (α) * dα.

d(ASIN · (α)) = dα/(1 - α ** 2)**(1/2).

d(ACOS · (α)) = - dα/(1 - α ** 2)**(1/2).

d(ATAN · (α)) = dα/(1 + α ** 2).

d(ACOT · (α)) = - dα/(1 + α ** 2).

d(ASEC · (α)) = dα/α * (1 - α ** 2)**(1/2).

d(ACSC · (α)) = -dα/α * (1 * α ** 2)**(1/2).

d(SINH · (α)) = COSH · (α) * dα.

d(COSH · (α)) = SINH · (α) * dα.

d(TANH · (α)) = SECH · (α) ** 2 * dα.

d(COTH · (α)) = - CSCH · (α) ** 2 * dα.

d(SECH · (α)) = - SECH · (α) * TANH · (α) * dα.

d(CSCH · (α)) = - CSCH · (α) * COTH · (α) * dα.

d(ASINH · (α)) = dα/(1 + α ** 2)**(1/2).

d(ACOSH · (α)) = - dα/(1 - α ** 2)**(1/2).

d(ATANH · (α)) = dα/(1 - α ** 2).

d(ACOTH · (α)) = dα/(1 - α ** 2).

d(ASECH · (α)) = - dα/(α * (1 - α ** 2)**(1/2)).

d(ACSCH · (α)) = - dα/(α * (1 + α ** 2)**(1/2)).

d(LOG · (α)) = dα/α.

d(EXP · (α)) = EXP · (α) * dα.

The application of this set of re-write rules to the matrix produced from the data vector constructs a second matrix designated as the derivative matrix. A subroutine, REPAR, will return the matrix to a fully parenthesized notation. However, without interruption the program will replace the original vector with the output of the derivative matrix, and upon repetition of the routines, produce successive derivatives. Test programs permit the vector to reach a maximum of 500 symbols in length. The number of derivatives of an expression, is, therefore, arbitrarily limited to this expansion.

Description of PAD

The re-write rules operate on the input matrix in such a manner that the derivative matrix becomes a continuation of the input matrix.

Let the first n rows of binary relations, $\theta(\alpha,\beta)$, represent the data vector, where $\theta_i(\alpha_i,\beta_i) \rightarrow R_j$.

For each $\theta_i(\alpha_i,\beta_i)$ the re-write rules will produce a row or a set of rows, $\theta_{n+j}(\alpha_{n+j},\beta_{n+j}), \ldots, \theta_{n+j+k}(\alpha_{n+j+k},\beta_{n+j+k})$.

Then each binary relation, $\theta_{n+i}(\alpha_{n+i},\beta_{n+i}) \rightarrow RD_k$, where the RD_k's are a new list of symbols added to the vocabulary of L to replace a row in the derivative matrix.

Finally, $\theta_{i+j}(RD_{n+i}, RD_{n+k})$ is the derivative of a function for which a single binary relation does not express the derivative generated by the definitions in the re-write rules.

For the example given in Chapter One:

INPUT MATRIX			DERIVATIVE MATRIX		
C	*	X	[C	*	1
SIN	·	R1	COS RD13	· *	R1 RD12
X	**	3	X 3	** *	2 RD15
A	*	R3	[A	*	RD16
R4	+	R2	[RD17	+	RD14
TAN	·	X	SEC RD19	· **	X 2
R6	**	2	R6 2 RD22	** * *	1 RD21 RD20
X	**	2	X 2	** *	1 RD24
B	*	R8	[B	*	RD25
R9	-	R7	[RD26	-	RD23
R10	/	R5	RD27 R10 RD28 RD30	* * - /	R5 RD18 RD29 RD31

The subroutine REPAR returns the derivative to a fully parenthesised expression by starting with the last row of the derivative matrix and replacing each R_k or RD_k encountered by its representation in the original row, that is, by elements of the sets VAR, CON, or FUN. At the same time, parentheses are replaced to preserve operator precedence.

In the following illustration the search tree for row replacement and parentheses insertion for the numerator of the previous example is diagrammed, with the operator for each binary relation placed at the tree nodes.

The scan begins with:

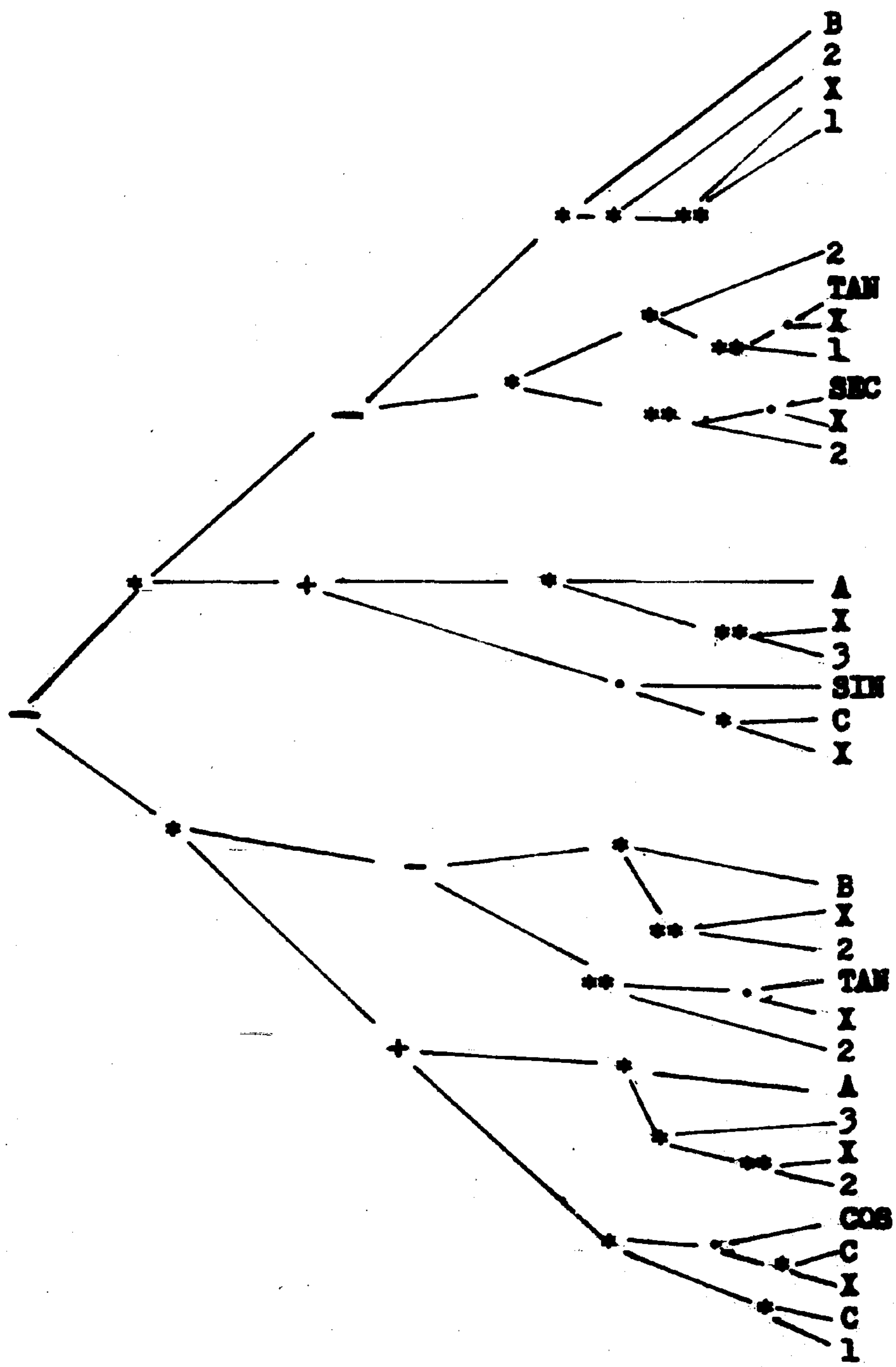
RD30 → RD26 - RD23

RD26 → B * RD25

RD25 → 2 * RD24

Its continuation can be easily followed through the tree. The leftmost negative sign is the operator for (RD26,RD23)

Search tree for row replacement and parentheses insertion for the numerator of Example 1.



The language of L is placed in the program by means of data statements. These in turn are stored in common so as to be available to the subroutines. In practice, the array names for the input matrix and the derivative matrix are distinct for the purpose of external identification, but the derivative matrix is, in fact, a continuation of the input matrix. The row numbers, however, do not overlap. Since the input matrix has an arbitrary number of rows, n, the program assigns the n+1 row to the first row of the derivative matrix. The array names for the input matrix and the derivative matrix are declared equivalent, but the above scheme always assigns them to separate sections of the block. An examination of the program shows that the equivalence statement simplifies the search in REPAR.

The subroutine EVAL provides for the numerical evaluation of the derivative. RAC, PAD, and REPAR manipulate symbols which are Hollerith constants. A new set of re-write rules are introduced which replace each binary relation in the derivative matrix with an equivalent arithmetic expression in Fortran, which can be evaluated for a given set of parameter, with the evaluation beginning with the first row and proceeding downward, since R_i may depend on some R_{i-j} rows, where i, j are positive integers and i is greater than j. When the last row is reached the value of the derivative has been calculated.

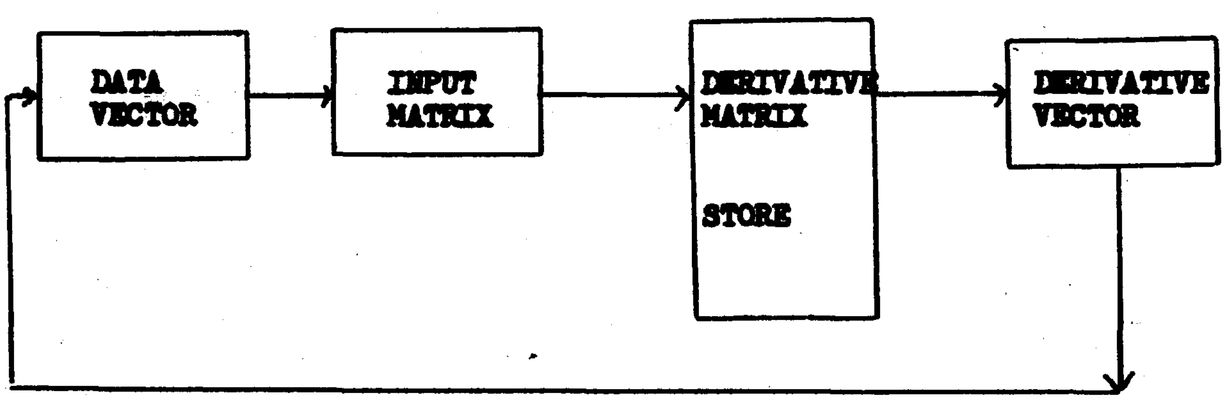
EVAL was developed, at first, as a checking routine, but it is possible to incorporate it in the pattern search. In Chapter Three we consider a "plane" pattern generated by each derivative, and the numerical evaluation could be considered as a point in this plane, but the variation introduced by the substitution of different parameters leads to procedures distinct from the type created by the use of symbols alone, and for this reason EVAL is not used in the pattern examination program.

Another subroutine to extend language L does prove to be useful. The subroutine SUBCON is called whenever a constant occurs in a data vector which is not identified in L. SUBCON replaces the constant with a Hollerith constant, and records the substitution. The new symbol is accepted by all routines in the pattern program.

CHAPTER THREE

Pattern Examination

In the previous chapters we have described the syntactic analysis of a data vector which leads to its representation as an n x 3 matrix whose row sequence preserves operator precedence. Similarly, a second set of re-write rules were chosen so that another n x 3 matrix was produced from the first, and represented a derivative of the data vector. By returning this production to a fully parenthesised expression, and allowing the latter to replace the first data vector, successive derivatives were obtained. We are not interested in the derivatives as such, but only in the successive patterns of the matrices representing them. This routine is regarded, now, as a pattern generating program which stores a sequence of expressions, thus:



Example 2 is inserted here for reference in the discussion of the computer program PEER which follows.

Example 2:

Let DATA = (A*X + B) **C, and n = the number of derivatives, where n $\leq$ C.

Then the input matrix is:

A	*	X
R1	+	B
R2	**	C

and the first four derivative matrices are:

(1)

A	*	1
C	-	1
R2	**	RD5
C	*	RD6
RD7	*	RD4

(2)

A	*	1
R1	-	1
R3	**	RD8
R1	*	RD9
RD10	*	RD7
C	*	RD11
RD12	*	A

(3)

A	*	1
R1	-	1
R3	**	RD11
R1	*	RD12
RD13	*	RD10
R6	*	RD14
RD15	*	A
RD16	*	A

(4)

A	*	1
R1	-	1
R3	**	RD14
R1	*	RD15
RD16	*	RD13
R8	*	RD17
RD18	*	A
RD19	*	A
RD20	*	A

Program PEER (Pattern Examination and Evaluation Routines) consists of three main parts: the pattern detector, the pattern evaluator, and the algorithm builder. In this section we will discuss the first of these.

The Pattern Detector

This part of the program accepts each derivative matrix and produces from it an m x 6 x n matrix, called GRAPH(I,J,K), where I ranges from 1 to m, m = the maximum index of R, J ranges from 1 to 6, and K from 1 to n, n = n^{th} index of the derivative.

For each derivative, K is fixed; therefore its GRAPH matrix is referred to as a "plane" pattern, and each new derivative produces a new plane. The set of planes forms a three dimensional representation of a changing pattern.

The formation of each plane proceeds as follows:

(1) For J = 1, let:

< BETA > : : = - | + | * | / | NEG | ** | .

Then GRAPH(I,1,K) = BETA(I) for $1 \leq I \leq m$, K fixed, where BETA(I) is an ordered sequence of operators, no adjacent pair of which are identical.

E.g. from Example 2:

For K = 1, BETA is the ordered set $\{*, \; -, \; **, \; *\}$.

(2) For J = 2, let:

TALLY $= \{1,2,3, \; . \; . \; . \; . \; , m\}$.

Then GRAPH(I,2,K) = TALLY(I), for $1 \leq I \leq m$, where TALLY(I) is an ordered sequence of the frequencies of successive repetions of each operator in BETA.

E.g. from Example 2:

For K = 1, TALLY = $\{1, \; 1, \; 1, \; 2\}$.

(3) For J = 3, let:

< ALP > : : = < CON > | < VAR > | < SUB > | (blank) ,

where CON and VAR are the symbols defined for the vocabulary L, and

< SUB > : : = AA | BB | CC | DD | EE | FF | GG | HH ,

that is, the replacement symbols used by the subroutine SUBCON for a constant (or variable) symbol not in L.

Then GRAPH(I,3,K) = ALP(I), $1 \leq I \leq m$, where ALP(I) is an ordered sequence of constant or variable symbols, or blank spaces.

E.g. from Example 2:

For K = 1, ALP = {A, C, __, C} .

(4) For J = 4, let:

RSEQ : : = {1, 2, 3, ... , m} .

Then GRAPH(I,4,K) = RESEQ(I), for $1 \leq I \leq m$, where RSEQ(I) is an ordered sequence of row indices.

E.g. from Example 2:

For K = 2, and J = 5,

ROWSEQ (I) = {1, 3, 1, 10, __, 12} .

(5) For J = 5, GRAPH(I,5,K) = ALP(I), and

(6) for J = 6, GRAPH(I,6,K) = RSEQ(I), in the same manner as for J = 3 and 4 respectively. When J = 3 or 4, GRAPH(I,3,K) and GRAPH(I,4,K) represents a scan of the first operands of the derivative matrix; when J is set at 5 or 6 the second operand is scanned.

Thus, $\theta\ (\ \alpha,\ \beta\ )$

$J = 1\ 2\quad 3\ 4\ 5\ 6$ are columns derived from

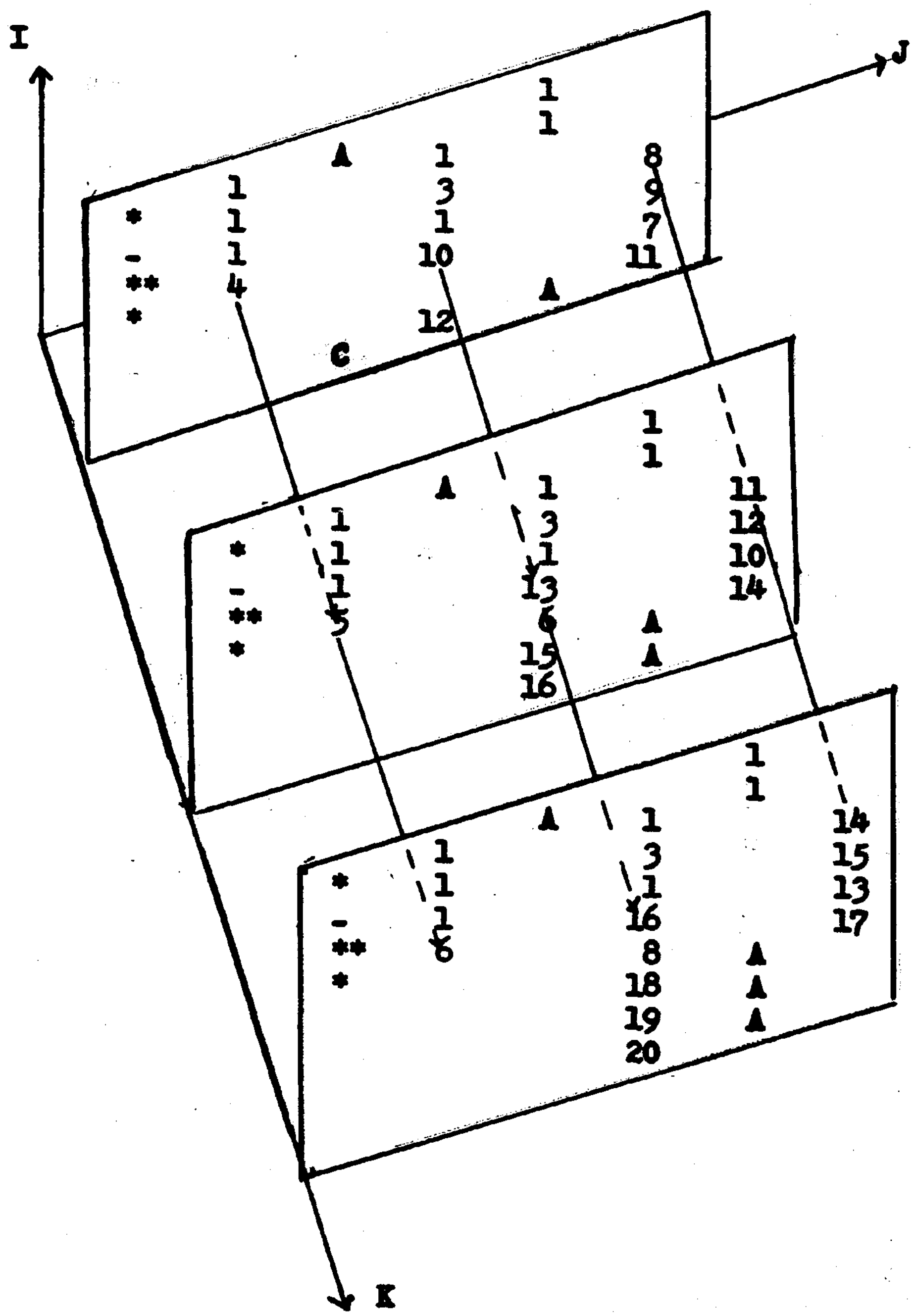
the scan of the binary relation in each row of the matrix.

For K = 1, GRAPH(I,J,K) in Example 2, the first pattern plane appears as:

*	1	A		1	
-	1	C		1	
**	1		2		5
*	2	C			6
			7		4

On page 33 three subsequent planes are shown for this same example. This diagram should be referred to in the discussion of searches in the "K" direction and the "I". A search on some points in the "K" direction is indicated.

Pattern Planes for K = 2, 3, 4 in Example 2.



The program is now ready to seek evidence of a pattern of change, if one exists. In constructing the GRAPH planes one pattern search was already made on each derivative. In this search the operator alphabet was determined with a frequency of successive repetitions assigned, thus forming a set of ordered pairs which completely describe the operator pattern. However, the construction of the GRAPH planes is straight-forward, and for the most part, a reorganization of the information in the derivative matrix. At this point, heuristic procedures enter, and the examination of the change in pattern is carried out by means of a series of trials which are evaluated in part two of PEER.

For frequencies, we may select a search for an arithmetic, geometric, Fibonacci, or any common mathematical sequence. However, in the problems taken from a freshman calculus text, no sequence other than an arithmetic sequence was found. For this reason it is the only type of numerical sequence tester that is incorporated permanently in the program. As can be seen from the construction of the GRAPH planes this sort of test is required for $J = 2,4,6$. In the first complete search as defined below, the arithmetic sequence test is basic. Alternates exist which may be called by the evaluating program discussed in the next chapter.

Actual trials on 50 expressions selected at random from a caluculus text indicated that it was efficient to make the first search a complete test of the pattern cube using a basic set of techniques selected from preliminary runs as the most frequently successful in pattern discovery. By a complete search is meant the application of this set to all rows and columns without using alternate routines. The decision to proceed to the algorithm builder in the case of a successful search, to stop, or to call alternate routines is the function of the second part of the program.

A complete search requires a single, two-directional test on each point of the plane. In the diagram on page 33, a one-directional test on three points is indicated. The nature of these tests is heuristically determined. The two which produced results on the above mentioned set of problems are as follows: we seek to establish that one of these alternatives, for successive planes holds:

(1) Operator alphabet is fixed or alternating

Operator frequencies are fixed or expand in an arithmetic sequence.

(2) Constants are fixed, or their repetition can be described by an arithmetic sequence.

Row sequence numbers change according to an arithmetic sequence in either or both K and I directions.

The simplest case for the operator alphabet is that of a fixed order with frequency expansion in an arithmetic sequence. Example 2 is an illustration of this. In the last section of this paper, on pp. 60-62, Example 3 shows an alternating and expanding pattern in the operator alphabet. In this example tests made on odd (or even) successive planes will yield a pattern.

The search is carried out in two directions. We begin with the K direction.

(1) Suppose BETA(I) = {*, -, **, *} for all K, and

TALLY(1) = {1, 1, 1, 2}

TALLY(2) = {1, 1, 1, 4}

TALLY(3) = {1, 1, 1, 5}

TALLY(4) = {1, 1, 1, 6}

which is the case for Example 2. BETA remains fixed for all planes examined, but the frequency tally on the last element becomes an arithmetic sequence after the second expansion.

Accordingly, the search for a pattern of change was built on a "sliding" routine which moved forward by dropping successively, K = 1, 2, . . . We will refer to this as <u>pattern advance</u>. In general, the test does not proceed beyond 10 planes, but the routine places no limit on the depth. In practice, a forward slide of two planes is usually sufficient as an indicator.

A different situation is observed in the pattern of operators in the input matrices of the same problem. It is sufficiently frequent and simple enough to warrant inclusion in the basic routines.

(2) Suppose BETA(1) = {*, +, **}

TALLY(1) = {1, 1, 1}

BETA(2) = {-, *, +, **, *}

TALLY(2) = {1, 1, 1, 1, 2}

BETA(3) = {-, *, +, **, -, *}

TALLY(3) = {1, 1, 1, 1, 1, 4}

BETA(4) = {- * + ** - *}

TALLY(4) = {1, 1, 1, 1, 2, 6}

Here the alphabet appears to vary, but a regular pattern of insertion can be detected. The recognition of this type of variation is handled by a routine which permits the expansion of the alphabetic vector, together with the pattern advance technique illustrated before. For discussion purposes we will refer to this combination as <u>pattern expansion</u>.

A failure to find an operator pattern could terminate the search. However, in accordance with the projected plan of making at least one complete test of all columns in GRAPH before reaching a decision node, no branching occurs until the end of the first scan.

A search in the I direction follows the search in the K direction:

(1) For J = 1, 3, 5, the alphabetic test for order and frequency is used. Blanks are delimiters in these columns; hence several short sequences may be discovered.

(2) For J = 2, 4, 6, the arithmetic sequence test is used, with zeros used as delimiters.

E.g. from Example 2:

For J = 4, the last k - 1 row indices form an arithmetic sequence.
For J = 5, BETA = A and TALLY forms an arithmetic sequence for the last k - 1 rows.

The combined search in the K and I directions results in a prediction of the elements in the derivative matrix of a selected n^{th} derivative. The construction of this matrix is discussed in Chapter Five.

CHAPTER FOUR

Pattern Evaluation

In the examination portion of PEER a set of routines acts on each point in a given plane which serves as a base. In this case, a "point" is one of the following: an operator symbol, a constant symbol, a number indicating either a frequency or a row. The pattern advance routine replaces this base with the k + 1 plane if necessary. All points are examined since it was decided that a complete set of routines would act on the entire pattern cube to some depth before attempting a decision. Obviously the examination set could be varied, but after a preliminary test on a random set of problems, a basic list was constructed.

The next objective is to evaluate the results of the search. Concurrently with the search, a decision matrix is constructed whose dimensions, I,J are identical with those of the K^{th} plane examined. The decision matrix is a logical array of 0's and 1's, where 0 indicates an unsuccessful search from this point, and 1 indicates a successful search.

In Example 2 the decision matrix for 4 planes is a 9×6 matrix of 1's, which indicates a high probability that a pattern has been detected. This complete set of 1's indicates that some pattern was found at each point beginning with $K = k$, $k \leq n - L$, where n is the depth of the search (the number of planes), and L is chosen arbitrarily.

Let G_k represent a point on a pattern plane for a fixed I, J.

(1) Then if for $K = k$, $G_k = G_{k+1} = G_{k+2} = , \ldots , = G_n$, the point $D_{i,j}$ in the decision matrix $= 1$.

(2) If for $K = k$, G_k, G_{k+1}, G_{k+2}, . . ., G_n is an arithmetic sequence (or any other sequence selected for the test set) then $D_{i,j} = 1$.

(3) If neither (1) nor (2) is true, then $D_{i,j} = 0$.

Let the value of $D_{I,J}$ equal the sum of all $D_{i,j}$'s in the matrix. From Example 2, we have:

$D = 54$, for $K = 4$, $k = 2$, $L = 1$.

$D = 53$, for $K = 4$, $k = 2$, $L = 2$.

$D = 51$, for $K = 4$, $k = 1$, $L = 3$.

In the last case, $k = 1$ is the minimum value, and $L = 3$ is the maximum value for $K = 4$; hence 51 is the minimum value of D. The very small change from the maximum value in the weight supports the hypothesis that a pattern of change exists.

The decision matrix is based entirely on a search in the K direction. The search made in the I direction is used for the purpose of algorithm building, and not to determine the existence of a pattern of change. It would be possible, by extending the number of rows and columns in the decision matrix, to include the information found in the I search, but no particular advantage appeared by doing this in any test problem.

A complete set of 1's causes Peer to accept the hypothesis that a pattern has been found. A single 0 causes it to reject this hypothesis and call for a new set of routines. An alternative to a second search set is the generation of additional pattern planes, but because a feasible number can be generated in the beginning this technique is not employed. A feasible number is based on the maximum symbol length arbitrarily selected.

CHAPTER FIVE

The Algorithm Builder

It is convenient to construct an auxiliary matrix at the same time that the examination and evaluation are in progress and to store it for use in algorithm building if the decision matrix causes the program to branch to this subroutine, as it will at the completion of a successful search.

PATCH (PATtern CHange) is a PEER routine which builds an auxiliary matrix, n x 6, which may be expanded to n+1, x10 to hold information from the I direction search. The first column contains the operator alphabet; the second column the difference, d_i, in their sequence expansion; the third and fifth columns are lists of constants; the fourth and sixth columns are the differences for the respective row number expansions. The n + 1 row is a symbol or difference indicating the nature of the column expansion in the same manner. If there is an inner change of shorter sequences in any column, 3 through 5, this is registered in columns 7 through 10 respectively, as before.

If the existence of a pattern of change is accepted as a hypothesis, PEER produces the projected change, that is, for each point of GRAPH(I,J,K), where K = n, it indicates:

Each point of GRAPH(I,J,K+L) is either fixed and identical with GRAPH(I,J,K), or has a pattern of change such that,

(1) the index of any row satisfies $i = k_i + (n-t)d_i$, where k_i is the first term of the sequence, d_i = the difference, t = k, the index of the first term, and n = the index of the derivative required.

(2) any constant $c = k_c + (n-t)d_c$, whose values are referenced as above.

Finally, the same procedure is employed for I + M rows of expansion called for by the I search.

The last routine, ALMA (ALgorithm MAtrix) builds a matrix for a given n^{th} derivative as a hypothesis. When the decision matrix has a maximum weight, the last plane used, the K-plane, is assigned as an expansion base. Using a routine REPOP, a new matrix is formed by expanding the K-plane according to the information in PATCH in the PEER routine. The program then returns to REPAR to output as a parenthesized expression the required derivative. Alternately, EVAL may be called for numerical evaluation, or it is possible to return to RAC if a succession derivatives after the n^{th} is desired.

The only test of the validity of the n^{th} derivative formed by ALMA was to use PAD to produce its counterpart by successive differentiation. On the test examples this was successful, whenever PEER was able to find a pattern, which establishes confidence in the methods employed.

As an algorithm builder this represents an approach suitable only to a machine, in that no single algebraic expression for an n^{th} sentence is produced, but instead there is a routine for jumping from k^{th} expression produced by the program to an n^{th} production required, if the particular representation created by the read program possesses a pattern of change that the program can recognize. The claim that this parallels the activity of the mathematician who generalizes on a symbol manipulation routine after some number of repetitions must, of course, be qualified.

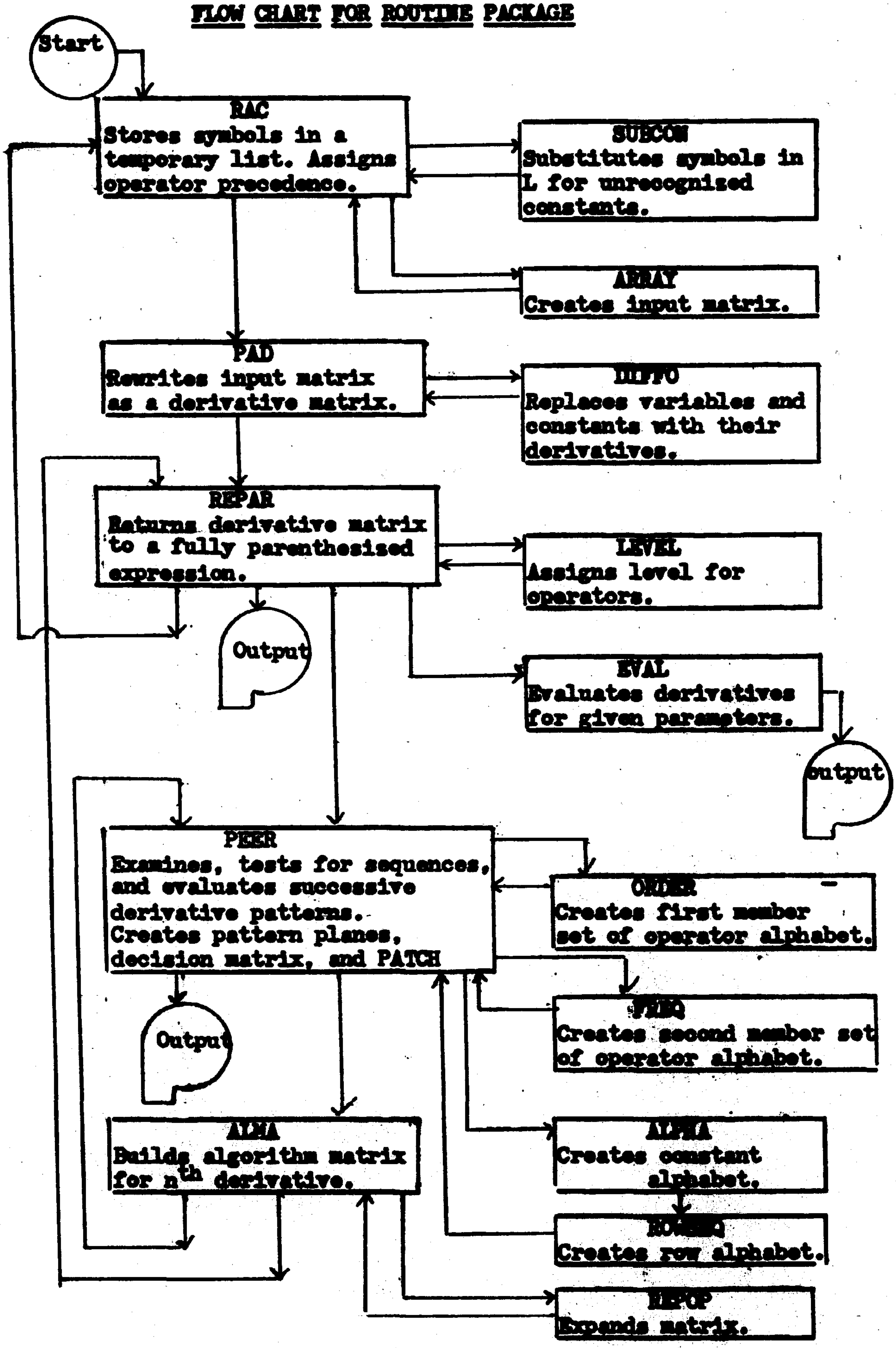
In their article on the simulation of human problem-solving [7], Bouricius and Keller remark:

> ". . . People customarily think at various levels of abstraction, and only rarely descend to the abstraction level of computer language. In fact, it seems that a large share of thinking is carried on by the equivalent of "subroutines" which normally operate on the subconscious level. It requires a good deal of introspection over a long period of time in order to dredge up these subroutines and simulate them. We believe that people assume that they know the logical steps they pursue when solving problems, primarily because of the fact that when two humans communicate, they do not need to descend to the lower levels of abstraction in order to explain to each other in a perfectly satisfactory way how they themselves solved a particular problem. The fact that they are likely to have very similar "subroutines" is obvious and also very pertinent. "

The subroutines employed by PEER are at a very low level of abstraction. Nevertheless, they do mechanize an inductive inference. They effect a "jump" from a k^{th} observation to an n^{th} observation, and in problems of the type tested in this experiment, this is accomplished in less than 30 seconds per problem.

Future experimentation could consist in refining some of the techniques. Improved editing procedures are needed to reduce the length of the matrices, for example. Enlarging the set of subroutines available to the pattern search is a more important extension. Application to other problem domains is certainly feasible, and may be capable of producing interesting results as a byproduct. It cannot be claimed, nor was it the purpose of the investigation, that the results obtained in the domain of analytic differentiation were important in themselves. We claim only that a form of mechanical inference was initiated, and that the method warrants extension.

FLOW CHART FOR ROUTINE PACKAGE



REFERENCES

1. Ershov, G. P. Programming program for the BESM computer, translated from the Russian by M. Nadler and edited by J. P. Cleave, Pergamon Press, London, New York, Paris, 1959.

also:

Beda, L.M., Karoev, L. N., Sukkekh, N. V. and Frolova, T.S., Programs for automatic differentiation for the machine BESM. Institute for Precise Mechanics and Computation Techniques, Academy of Science, USSR, Moscow, 1959.

2. Hanson, J. W. Analytic Differentiation by Computer. Computation Center, University of North Carolina, December 1961. Private communication with the author.

also:

Hanson, J.W., Caviness, J., Joseph, Camilla, Analytic Differentiation by computer, ACM, June 1962, pp. 349-355.

3. Arden, B.W. An introduction to digital computing. Adison-Wesley, Reading, Massachusetts, 1963.

4. Potts, D. H. Algorithms for proving theorems: a survey. Grant no. GP-536 NSF. November 1963, Computer Center, University of California, San Diego, La Jolla, California.

5. Robinson, J. A. A machine-oriented logic based on the resolution principle. U. S. Atomic Energy Commission. Argonne National Laboratory and Rice University, August, 1963.

6. Backus, J. W. The syntax and semantics of the proposed international language of the Zurich ACM - Gamm conference. Proceedings of the International Conference on Information Processing, UNESCO, 1959, pp. 125,132.

7. Bouricius, W. G. and Keller, J. M. Simulation of human problem-solving. Joint Computer Conference. Proceedings v. 15 (Western), March 1959, pp. 116-119.

BIBLIOGRAPHY

Abramson, N., Braverman, D. Learning to recognize patterns in a random environment. IRE transactions on information theory, v. IT-8, no. 5, 1962, pp. 58-63.

Amoroso, S. An application of heuristic programming to the problem of theorem proving by machine. Army Electronics Research and Development Agency, March 1963.

Andrew, A. M. Learning machines. Mechanization of thought processes; proceedings of a symposium held at the National Physical Laboratory, November 1958. London, H. M. Stationery Office, 1959. v. 1, pp. 473-509.

Arnold, R. F. A compiler capable of learning. Joint Computer Conference. Proceedings. v. 15, (Western) March 1959, pp. 137-143.

Campaigne, H. Some experiments in machine learning. Joint Computer Conference. Proceedings, v.16,(Western) March 1959, pp. 173-175.

Carr, J. W. Recursive subscripting compilers and list-type memories. Communications of the ACM, v. 2, no. 2, February 1959, pp. 4-6.

Engelhart, D. C. Augmenting human intellect: a conceptual framework. Stanford Research Institute, October 1962.

Fischler, M., and others. An approach to general pattern recognition. IRE transactions on information theory, v. IT-8, no. 5, September 1962, pp. 64-73.

Fogel, L. J. Toward inductive inference automata. International Conference on Information Processing. 2d., Munich, 1962; proceedings of IFIP Congress 62. Editor: Cicely M Popplewell. Amsterdam, North-Holland Publishing Co., 1963, pp. 395-400.

Gelernter, H. A note on syntactic symmetry and the manipulation of formal systems by machines. Information and Control, v.2, April 1959, pp. 80-89.

George, F. H. Inductive machines and the problem of learning. Cybernetica, v. 2, 1959, pp. 109-126.

Green, B. F. Computer languages for symbol manipulation. IRE transactions on human factors in electronics, v. HFE-2, March 1961, pp. 3-8.

Greene, P. H. A suggested model for information representation in a computer that perceives, learns, and reasons. Joint Computer Conference. Proceedings, v. 17,(Western) May 1960, pp. 151-164.

Hagelbarger, D. W. SEER, a sequence extrapolating robot. IRE transactions on electronic computers, v.EC-5, March 1956, pp. 1-7.

Johnson, D. L. and Kobler, A. L. Man-computer interface study. Seattle, Washington University College of Engineering. June 1963. 155 p.

Karush, W. Mathematical programming, man-computer search and system control. Santa Monica, California, System Development Corporation, May 1962.

Keller, J. M. and Bouricius, W. G. Simulation of human problem-solving. Joint Computer Conference. Proceedings v. 15(Western) March 1959, pp. 116-119.

Kilburn, T., and Grimsdale, R. L. Experiments in machine learning and thinking. International Conference on Information processing, Paris, 1959. Information processing: proceedings, Paris, UNESCO, 1960. pp. 303-309.

Laemmel, A. E. Machine learning processes applied to pattern recognition. Polytechnic Institute of Brooklyn, December 14, 1961.

McCarthy, J. Recursive functions of symbolic expressions and their computation by machine. Communications of the ACM. v.3 no.4, April 1960, pp.184-195.

Marill, T. A note on pattern-recognition techniques and game-playing programs. Information and control, v.6, September 1963, pp. 213-217.

Marill, T. and Green, D. M. Statistical recognition functions and the design of pattern recognizers. IRE transactions on electronic computers, v. EC-9, December, 1960, pp. 472-477.

Metzelaar, P. Mechanical realization of pattern recognition. Bionics Symposium. 1st Dayton, Ohio, 1960. WADD technical report 60-600. pp. 419-435.

Newell, A., Shaw, J.C. , Simon, H. A. The processes of creative thinking. Santa Monica, Rand Corporation, 1958. Rand Corporation Paper, P-1320.

Perlis, A. J. and Thornton, C. Symbol manipulation by threaded lists. Communications of the ACM, v. 3, no. 4, April 1960, pp. 195-204.

Putter, P.S. Zur theorie der Denkvorgange. Electronische Datenverarbeitung. v.5, February 1963. pp. 27-40; April 1963, pp. 88-101; June 1963, pp. 123-132.

Sakaguchi, M. Information pattern, learning structure, and optimal decision rule. Information and control. v. 6, September 1963, pp. 218-229.

Sebestyen, G. S. Decision-making processes in pattern recognition. New York, Macmillan, 1962.

Simon, H. A. Experiments with a heuristic compiler. Association for Computing Machinery. National Conference. Digest of technical papers, New York, L. Winner, 1962.

Slagle, J. R. A heuristic program that solves symbolic integration problems in freshman calculus (SAINT) Cambridge, Mass., 1961. Thesis-MIT.

Solomonoff, R. J. A preliminary report on a general theory of inductive inference. Us. S. Air Force Office of Scientific Research. Cambridge, Mass., Zator Co., November 1960.

Solomonoff, R. J. Research in inductive inference; Progress report for April 1959--Nov. 1960. Cambridge, Mass., Zator Company., January 1961.

Travis, L. E. Observing how humans make mistakes to discover how to get computers to do likewise. Santa Monica, Calif., System Development Corporation, 1962.

Uhr, L. and Vossler, C. A pattern recognition program that generates, evaluates, and adjusts its own operators. Joint Computer Conference. Proceedings, v. 19 (Western) May 1961, pp. 550-570.

Unger, S. H. GIT--a heuristic program for testing pairs of directed line graphs for isomorphism. Communications of the ACM, v. 7, January 1964, pp.26-34.

Wang, H. Proving theorems by pattern recognition. Communications of the ACM, v.3, April 1960, pp. 220-234.

Wigington, R. L. A machine organization for a general list processor. IEEE transactions on electronic computers, v.EC-12, December 1963, pp. 707-714.

EXAMPLE 1

DATA VECTOR

$	(	B	*	X	**	2	-	(	TAN
.	X	)	**	2	)	/	(	A	*
X	**	3	+	SIN	.	(	C	*	X
)	)								

INPUT MATRIX

C	*	X
SIN	.	R1
X	**	3
A	*	R3
R4	+	R2
TAN	.	X
R6	**	2
X	**	2
B	*	R8
R9	-	R7
R10	/	R5

DERIVATIVE MATRIX

C	*	1
COS	.	R1
RD13	*	RD12
X	**	2
3	*	RD15
A	*	RD16
RD17	+	RD14
SEC	.	X
RD19	**	2
R6	**	1
2	*	RD21
RD22	*	RD20
X	**	1
2	*	RD24
B	*	RD25
RD26	-	RD23
RD27	*	R5
R10	*	RD18
RD28	-	RD29
R5	**	2
RD30	/	RD31

DERIVATIVE VECTOR

$	(	(	(	(	B	*	(	2	*
X	**	1	)	)	-	(	(	2	*
TAN	.	X	**	1	)	*	SEC	.	X
**	2	)	)	*	(	A	*	X	**
3	)	+	SIN	.	(	C	*	X	)
)	-	(	(	(	B	*	X	**	2
)	-	(	TAN	.	X	**	2	)	)
*	(	A	*	(	3	*	X	**	2
)	)	+	(	COS	.	(	C	*	~~X~~
)	*	(	C	*	1	)	)	)	)
/	(	(	A	*	X	**	3	)	+
SIN	.	(	C	*	X	)	**	2	)

ANOTHER EXAMPLE FROM PAD

DATA VECTOR

$	(	SIN	•	X	-	ATAN	•	X	)
/	(	X	**	2	*	LOG	•	(	1
+	X	)							

INPUT MATRIX

1	+	X
LOG	•	R1
X	**	2
R3	*	R2
ATAN	•	X
SIN	•	X
R6	-	R5
R7	/	R4

DERIVATIVE MATRIX

0	NEG	1
R1	**	RD9
X	**	1
2	*	RD11
RD12	*	R2
RD10	*	R3
RD13	+	RD14
X	**	2
1	+	RD16
1	/	RD17
COS	•	X
RD19	-	RD18
RD20	*	R4
R7	*	RD15
RD21	-	RD22
R4	**	2
RD23	/	RD24

DERIVATIVE VECTOR

$	(	COS	•	X	-	(	1	/	1
+	(	X	**	2	)	)	)	*	X
**	2	*	LOG	•	(	1	+	X	)
-	(	SIN	•	X	-	ATAN	•	X	)
*	(	2	*	X	**	1	*	LOG	•
(	1	+	X	)	)	+	(	1	+
X	**	(	0	-	1	)	*	X	**
2	)	/	(	X	**	2	*	LOG	•
(	1	+	X	)	)	**	2		

Examples of successive differentiation: (Example 3)

DATA VECTOR

```
$    COSH .   (   X   **   2   -   3   *   X   +   1   )
```

INPUT MATRIX

```
3      *    X
X      **   2
R2     -    R1
R3     +    1
COSH   .    R4
```

DERIVATIVE MATRIX

```
3      *    1
X      **   1
2      *    RD7
RD8    -    RD6
SINH   .    R4
RD10   *    RD9
```

DERIVATIVE VECTOR

```
$    SINH  .   (   (   X   **   2   -   3
*    X     )   +   1   )   *    (   2   *
X    **    1   -   3   *   1    )
```

INPUT MATRIX

```
3      *    X
X      **   2
R2     -    R1
R3     +    1
SINH   .    R4
2      *    X
R6     -    3
R7     *    R5
```

DERIVATIVE MATRIX

3	*	1
X	**	1
2	*	RD10
RD11	-	RD9
COSH	.	R4
RD13	*	RD12
2	*	1
RD15	*	R5
RD14	*	R7
RD16	+	RD17

DERIVATIVE VECTOR

$	(	2	*	1	*	SINH	.	(	(
X	**	2	-	3	*	X	)	+	1
)	)	+	(	COSH	.	(	(	X	**
2	-	3	*	X	)	+	1	)	*
(	2	*	X	**	1	-	3	*	1
)	*	(	2	*	X	-	3	)	)

INPUT MATRIX

3	*	X
X	**	2
R2	-	R1
R3	+	1
COSH	.	R4
2	*	X
R6	-	3
R7	**	2
R8	*	R5
3	*	X
X	**	2
R11	-	R10
R12	+	1
SINH	.	R13
2	*	R14
R15	+	R9

DERIVATIVE MATRIX .

3	*	1
X	**	1
2	*	RD18
RD19	-	RD17
SINH	.	R4
RD21	*	RD20
2	*	1
R7	**	1
2	*	RD24
RD25	*	RD23
RD26	*	R5
RD22	*	R8
RD27	+	RD28
3	*	1
X	**	1
2	*	RD31
RD32	-	RD30
COSH	.	R13
RD34	*	RD33
2	*	RD35
RD36	+	RD29

DERIVATIVE VECTOR

$	(	2	*	COSH	.	(	(	X	**
2	-	3	*	X	)	+	1	)	*
(	2	*	X	**	1	-	3	*	1
)	)	+	(	2	*	(	2	*	X
-	3	)	**	1	*	2	*	1	*
COSH	.	(	(	X	**	2	-	3	*
X	)	+	1	)	)	+	(	SINH	.
(	(	X	**	2	-	3	*	X	)
+	1	)	*	(	2	*	X	**	1
-	3	*	1	)	*	(	2	*	X
-	3	)	**	2	)				

EXAMPLE 2

DATA VECTOR

$	(	A	*	X	+	B	)	**	C

INPUT MATRIX

A	*	X
R1	+	B
R2	**	C

DERIVATIVE MATRIX

A	*	1
C	-	1
R2	**	RD5
C	*	RD6
RD7	*	RD4

DERIVATIVE VECTOR

$	C	*	(	A	*	X	+	B	)
**	(	C	-	1	)	*	A	*	1

INPUT MATRIX

C	-	1
A	*	X
R2	+	B
R3	**	R1
C	*	R4
R5	*	A

DERIVATIVE MATRIX

A	*	1
R1	-	1
R3	**	RD8
R1	*	RD9
RD10	*	RD7
C	*	RD11
RD12	*	A

DERIVATIVE VECTOR

$	C	*	(	C	-	1	)	*	(
A	*	X	+	B	)	**	(	(	C
-	1	)	-	1	)	*	A	*	1
*	A								

INPUT MATRIX

C	-	2
A	*	X
R2	+	B
R3	**	R1
C	-	1
C	*	R5
R6	*	R4
R7	*	A
R8	*	A

DERIVATIVE MATRIX

A	*	1
R1	-	1
R3	**	RD11
R1	*	RD12
RD13	*	RD10
R6	*	RD14
RD15	*	A
RD16	*	A

DERIVATIVE VECTOR

$	C	*	(	C	-	1	)	*	(
C	-	2	)	*	(	A	*	X	+
B	)	**	(	(	C	-	2	)	-
1	)	*	A	*	1	*	A	*	A

INPUT MATRIX

C	-	3
A	*	X
R2	+	B
R3	**	R1
C	-	2
C	-	1
C	*	R6
R7	*	R5
R8	*	R4
R9	*	A
R10	*	A
R11	*	A

DERIVATIVE MATRIX

A	*	1
R1	-	1
R3	**	RD14
R1	*	RD15
RD16	*	RD13
R8	*	RD17
RD18	*	A
RD19	*	A
RD20	*	A

DERIVATIVE VECTOR

$	C	*	(	C	-	1	)	*	(
C	-	2	)	*	(	C	-	3	)
*	(	A	*	X	+	B	)	**	(
(	C	-	3	)	-	1	)	*	A
*	1	*	A	*	A	*	A		

PATTERN PLANES

K = 4

*	1A	01	0
-	1	11	0
**	1	3	14
*	6	1	15
	0	16	13
	0	8	17
	0	18A	0
	0	19A	0
	0	20A	0

PATCH

*	0A	01	0
-	0	01	0
**	0	0	3
*	1	0	3
		3	3
		2	3
		3A	0
		3A	0
		3A	0
		1A	0

K = N = 10

A	*	1
RD1	-	1
RD3	**	RD32
RD1	*	RD33
RD34	*	RD31
RD20	*	RD35
RD36	*	A
RD37	*	A
RD38	*	A
RD39	*	A
RD40	*	A
RD41	*	A
RD42	*	A
RD43	*	A
RD44	*	A

TITLE OF THESIS INDUCTIVE INFERENCE ON COMPUTER GENERATED PATTERNS

Full Name Sister Mary Kenneth Keller, BVM

Place and Date of Birth Cleveland, Ohio Dec. 17, 1913

Elementary and Secondary Education

Cleveland and Chicago Parochial Schools

Immaculata High School, Chicago, Illinois

Colleges and Universities: Years attended and degrees

Clarke College 1933-35 Mundelein College 1935

De Paul University, Chicago 1942-1952 A.B. 1943, M.S. 1952

Purdue University 1957 Dartmouth College 1961

University of Wisconsin 1962-65 Ph.D. June 1965

Membership in Learned or Honorary Societies Society for Industrial and Applied Mathematics

Association for Computing Machinery ,

The Mathematical Association of America

Publications

Major Department Computer Sciences

Minor(s) Mathematics

Date May 21, 1965

Signed Preston C Hammer

Professor in charge of thesis