SYMBOLIC MATHEMATICAL LABORATORY

by

WILLIAM ARTHUR MARTIN

B.S., Massachusetts Institute of Technology
1962

M.S., Massachusetts Institute of Technology
1962

SUBMITTED IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
January, 1967

Signature of Author . William A. Martin . . . . .
Department of Electrical Engineering, January 9, 1967

Certified by . . Marvin Minsky . . . .
Thesis Supervisor

Accepted by . . Truman S. Gray . . . .
Chairman, Departmental Committee on Graduate Students

SYMBOLIC MATHEMATICAL LABORATORY

by

WILLIAM ARTHUR MARTIN

Submitted to the Department of Electrical Engineering on January 9, 1967 in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

ABSTRACT

A large computer program has been developed to aid applied mathematicians in the solution of problems in non-numerical analysis which involve tedious manipulations of mathematical expressions. The mathematician uses typed commands and a light pen to direct the computer in the application of mathematical transformations; the intermediate results are displayed in standard text-book format so that the system user can decide the next step in the problem solution.

Three problems selected from the literature have been solved to illustrate the use of the system. A detailed analysis of the problems of input, transformation, and display of mathematical expressions is also presented.

Thesis Supervisor: Marvin L. Minsky

Title: Professor of Electrical Engineering

ACKNOWLEDGEMENTS

The work herein was supported in part by the M.I.T. Computation Center, and in part by Project MAC, an M.I.T. research program sponsored by the Advanced Research Projects Agency, Department of Defense, under Office of Naval Research Contract number Nonr-4102(01). Reproduction in whole or in part is permitted for any purpose of the United States Government.

The author expresses his gratitude to Marvin Minsky for his supervision of this thesis and to the directors of Project MAC for providing a sympathetic environment where the work could be carried out. I also wish to thank Herbert Teager and Oliver Selfridge for reading the manuscript and Tim Hart for his assistance when I first joined the Artificial Intelligence Group.

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

TABLE OF CONTENTS (continued)

TABLE OF CONTENTS (continued)

Chapter I

INTRODUCTION

The purpose of this thesis is to advance the computer programming technology required for the real time solution of non-numerical problems in applied mathematics. Generally, the first use of computers in any area has been the mechanization of the routine parts of tasks in essentially the same manner as they were formerly done by hand. Later, when the computer resources are better understood, the methods of doing these tasks are altered to make better use of the computer. J. C. R. Licklider(2) has described the "man-computer symbiosis" which would result from the most economical utilization of the abilities of man and currently available machines in a problem solving situation. "Men will make approximate and fallible, but leading contributions, and they will define criteria and serve as evaluators, judging the contributions of the equipment and guiding the general line of thought. In addition, men will handle the very low probability situations when such situations arise. ...The information-processing equipment, for its part, will convert hypotheses into testable models and then test the models against data." Licklider stated further that the equipment will simulate, transform, interpolate, extrapolate, carry out routinizable clerical operations, and remember precise values and exact details.

This degree of man-machine cooperation has been and will continue to be approached by stages. This is because it is very difficult to determine the many small features necessary to make workable a system that differs greatly from those currently in use, and because mathematicians will not use a system which is not clearly more convenient than their tried and true aids, such as pencil and paper. Furthermore, considerable research will be required to determine just how much of mathematics is routinizable. One might envision four roles which the computer could take, each requiring increased responsibility.

The first step would be to help the mathematician overcome straightforward drudgery. This is often done by programming the simple iterative operations needed in straightforward problems in numerical analysis. <u>A second goal would be to help the mathematician work in areas that he understands in principle, but where he does not have the technical facility, gained from long practice and accuracy, which frees him to focus on the creative aspects of the problem</u>. Such a skill is required, for example, for the approximate solution of differential equations using series expansion techniques; <u>the programming system constructed for this thesis operates at this level</u>. A third goal would be to help the mathematician in areas where he is only a beginner--where he knows the names and purposes of the algorithms, but does not know their details. The computer must know and apply criteria which determine what methods should be used on a given example. Some current research on symbolic integration seems to be a step toward systems of this type.(8) Research in computer problem-solving tactics for the use of fixed resources on specific tasks would be useful for systems at this level. A fourth goal would be the construction of an almost autonomous problem solver; the machine would help formulate the general strategy of the problem solution. It is at this level that any productive research in computer theorem-proving strategy would be most applicable to a working system.

Of course, to build a working system which represents a significant advance toward these goals requires a great deal of research in several areas, often too much for any one project. It is difficult to work much in advance of the general level of programming technology without having to do a lot of ground work, which only builds tools for the project at hand. To develop the system in this thesis, which realizes in the laboratory the second of the above goals, it was necessary to set up systems programs and a programming language environment for real-time interaction with large symbolic programs. It required new techniques for input and output of mathematical expressions. Mathematical expressions were represented as data in the computer, and we had to investigate algorithms needed to transform them. Specific problem areas where mathematicians could most benefit from such a system had to be defined. Working in so many areas, only some of the possibilities could be explored and a fixed strategy with respect to the trade-offs between programmer effort, program efficiency and usefulness, and the number of new ideas had to be followed.

The particular emphasis of the thesis can best be understood through a review of work developed concurrently with it. A bibliography of the work in the field has been prepared by Sammet (1), and no attempt will be made to mention every paper. In the sections to follow some research areas will be listed. Then the different research projects carried out by other people will be discussed, grouped according to their area of emphasis. In conclusion, the choice of research for this thesis will be stated.

To better use the computer as an aid to the mathematician, there are several lines of research which can be pursued, each affording the possibility of doing something new without overreaching for currently unrealistic levels of overall performance. These are as follows:

1. The provision of computer capability for solving existing problems in mathematics. Special programs might be written for a particular problem. The programs would be simple, but the problem solution would be new.

2. The development of complex programming structures and languages suitable for more complex mathematical algorithms.

3. The development of input-output techniques to bring the mathematician closer to the computer.

4. The discovery of procedures for mechanizing the more routine mathematical transformations.

5. The development of methods for solving problems which are feasible only with computer implementation.

6. The extension of mathematical rigor to computer procedures and languages in the context of mathematical procedures.

There is naturally some difference of opinion as to the usefulness, significance, and feasibility of research in the different areas, and over the past five years more than fifteen people and projects have taken their stand.

Several programs have been written in the LISP 1.5 language, which was developed by the M.I.T. Artificial Intelligence Group.(19) LISP 1.5 provides both numerical and list structure data and automatic storage allocation. In addition, the specification of algorithms as recursive functions is particularly suitable for operating on the elementary functions in analysis, which can be represented as a tree structure of subexpressions. To demonstrate the capabilities of LISP, Hart(3) wrote a program to put the elementary functions in a simplified form. He had to define a canonical order for expressions representing the elementary functions and he found that not only were some simplifications very hard to program, but that the use of an expression determined its "simplest" form. Carrying on this work, Russell and Wooldridge(4) at Stanford, also interested in complicated LISP programs, added new features to cope with the inefficiencies and shortcomings of Hart's program. They introduced user options, such as expansion of products, and special representation for polynomials. They added routines for differentiation. They had ideas for discovering subexpressions such as $\sin^2 x + \cos^2 x$ and methods to avoid simplifying identical subexpressions more than once, but they quit working and the program was not improved further, probably because each change required many small revisions. A new simplification program was written by Korsvold.(5) He reduced the amount of fixed program structure and implemented a way for adding simplification rules in the form of patterns containing a free variable. He had a complicated strategy for applying the rules, but the matching of the rule to the data only covered simple cases. At the same time, Fenichel(6) at Harvard recognized the importance of an explicit body of rules; this body of rules would provide program flexibility. In contrast to the others, Fenichel emphasized the formal power and structure of his program instead of showing how his implementation was an efficient model for the data and algorithms. He emphasized the idea of computation by local transformations.

There are two more examples of complex programs for mathematical expression transformation. While Hart was writing the first of the programs for algebraic simplification, Slagle(7) wrote a program which solved freshman calculus integration problems. Although Slagle primarily wanted to show how heuristics and a judicious choice of subproblems could be used to solve examples in an area for which a general algorithm was not known, he also had to find properties of mathematical expressions which would be useful in selecting a trial transformation. Thus, he wrote a complicated routine for matching mathematical expressions against a pattern. Slagle's program demonstrated a working scheme for the heuristic selection of subgoals and provided food for thought about executive structures, but it misled some people to think that few integration problems could be solved algorithmically. Recently Moses has increased the range of examples which can be integrated with algorithms. Moses(8) has continued the work on pattern matching routines.

Concurrent with this work was the work of those who took a skeptical view of the possibility of creating complex programs which were efficient enough for immediate use. The ALPAK system(9), developed at Bell Telephone Laboratories, is a polynomial manipulation package which has been used to solve problems for the research workers there. It is so much more efficient than the systems above, that it provides a definite incentive to produce optimum coding for common classes of problems. In addition, it provides mathematicians with a useful facility, but it is limited to problems which can be solved by polynomial manipulation. An excellent polynomial manipulation package has also been developed by Collins(10) at I.B.M. In his paper, he gives a good discussion of the relative efficiency of different systems. The FORMAC language(11), also developed at I.B.M., is intended for the efficient solution of problems requiring algebraic manipulation. FORMAC programs are similar to FORTRAN programs and require statements which are more familiar to programmers than to mathematicians, such as the declaration of symbolic variables and the direction of input-output. The language is not as flexible as the LISP systems, but it is faster. Another language is FORMULA ALGOL(12), which is being developed at Carnegie Tech. It is a version of ALGOL which contains list structure and mathematical expression data types and a built-in pattern matching facility. It contains numerous design decisions, such as the use of an addition operator with only two arguments, which seem controversial.

During this same period, better man-machine communication was developed in certain systems for real-time interaction with programs for numerical analysis. One of the first programs was the JOSS system, developed at RAND in 1963.(13) The fact that it was implemented on a small machine limited its capability to simple calculations, but it stands as an example of the attention to details of "human engineering" which makes a system pleasant to use. Culler(14) demonstrated the real-time solution of numerical problems by methods which could not be used in a closed shop computer facility, since they required frequent man-machine interaction. Culler introduced the graphical display of 127 points approximating a function to be transformed numerically. He attempted to use special keys on a user console to simplify the input of expressions and commands. The primatives of Culler's system are very elementary; the user must first combine these operations into programs which perform the transformation he needs.

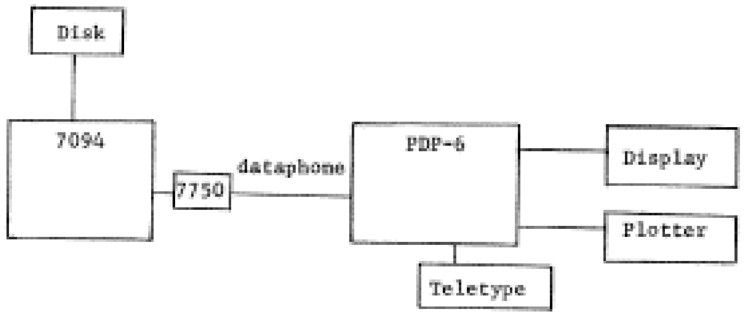
On-line numerical analysis is also performed my MAP(15), which operates at Project MAC. MAP contains error-checking and comments to guide the user as he learns. It also establishes a set of conventions for the addition of new subroutines, so that they can be used conveniently. MATHLAB(16) is the first working program for on-line non-numerical analysis. It is being developed at the MITRE Corporation with computer support from Project MAC. The initial version contained the Russell-Wooldridge simplification program. In addition to the routines for expression expansion and differentiation, there were routines for factoring and integration. There are many differences, but with the exception of the factoring and integration routines, the initial version of MATHLAB was essentially a subsystem of the system described here.

There has been only a little work on the mathematics associated with algebraic symbol manipulation. Several programs have been written to accomplish a specific task which requires mathematical manipulation, but these are rather simple from a programming point of view, and there have not been enough of them to change the applied mathematician's concept of the resources available to him. Therefore, they have not resulted in much new mathematics.

To improve the efficiency of his polynomial manipulation package, Collins developed a new greatest common divisor algorithm for polynomials with integer coefficients. Moses has investigated algroithms for solving systems of polynomial equations and for integration. Manove(17) has mechanized the integration of rational functions, employing a program for the factorization of polynomials in several variables over the integers written by Bloom(17). Engelman(17) is expanding his MATHLAB through the inclusion of these routines as well as others that have been written for the manipulation of matrices, direct and inverse Laplace transforms, the solution of equations, and the typewriter display and editing of mathematical expressions. An interesting theoretical result was obtained by Richardson(18), who recently proved that the questions of whether (i) an elementary function is identically zero or (ii) is integrable in closed form, are recursively unsolvable.

The mathematical laboratory described in this thesis is very complex and explores a number of new ideas about hash coding, descriptive properties of mathematical expressions, and system organization. An examination is made of the types of programming structures needed in non-numerical analysis. Some very complicated routines for expression analysis and display are explained. On the other hand, no new mathematical algorithms are introduced and the programs are explained without mathematical formalism. As opposed to the other complex LISP systems, the system was constructed to be immediately useful as an aid to applied mathematicians in the solution of certain problems. As opposed to some of the other problem solving systems, it does not solve these problems in an economical manner. Instead, attention has been concentrated on providing a rich enough notation for input and display and a great enough variety of mathematical transformations to achieve better man-computer symbiosis during the problem solving procedure. As a demonstration of this man-machine interaction, we will present "scripts" showing the solution of some non-trivial problems in applied mathematics.

The current hardware configuration for the mathematical laboratory is shown in Figure 1.



The system is quite inefficient, in terms of the amount of the total computation time required by the slow transmission of information over the dataphone, the frequent disk accesses necessitated by the small 7094 core memory, and the fact that many of the routines run interpretively. These inefficiencies could be eliminated by introducing the entire system into the PDP-6, with the large core memory now available.

The configuration in Figure 1 was used initially as it was the only means then available for obtaining the minimum requirements for the system: a time-shared machine with a large memory connected to a display with fast light pen response. The use of a peripheral computer for input has the disadvantage that the input signals cannot be intrepreted by the peripheral computer in terms of the whole data base, which is in the main machine. This limits the immediate feedback which can be used to simplify the user's input task. In addition, the use of two machines complicates the day-to-day routine required to develop a system. The justification for two machines would be the more economical assignment of the different types of computation to be done, or the need to have the user located at a terminal remote from the main machine.

Chapter II

A STEP BY STEP SOLUTION OF THREE PROBLEMS IN NON-NUMERICAL ANALYSIS

The mathematical laboratory consists of a PDP-6 computer linked by dataphone with the Project MAC time sharing system. When the laboratory is in operation, versions of the LISP programming language are running in both the PDP-6 computer and the Project MAC 7094 computer. The user communicates with the PDP-6, which is used only for input-output, by means of a teletype, a DEC Type 30 display, a light pen, and a Calcomp plotter. The PDP-6 relays messages to and from the Project MAC time sharing system, where transformation routines written in the LISP language are applied to the mathematical expressions. Mathematical expressions can be displayed and plotted in standard textbook format. Input of expressions and commands is from the teletype, but subexpressions of displayed expressions can be referenced during input. If the light pen is pointed at the main connective of a subexpression, the subexpression can now be referenced in the typed input string. The exact format of the input commands and a brief description of the available mathematical transformations will be presented in the next section. This description is followed by the step-by-step solution of three problems in applied mathematics. The details of the programs will be presented in later chapters.

The Mathematical Operators

The commands typed at the PDP-6 are similar to Algol statements. The commands are typed and executed one at a time. More complex operations involving the definition and alteration of commands and the introduction of more pneumonics and man machine interaction will be described later. The commands consist of infix operators, functions, and variables. Functions and variables can be subscripted and any subexpression can be quoted. A sample command which requires most of the notation is:

```
    #'E1←'(X + Y) * 'DRV(':T,1,DRV('U,2,E1)) + !E2,20 + '(F[I,J](X,Y))↑2#
```

In words: the name E1 is assigned to the expression which is the sum of three terms. The first term is the product of (X + Y) with the unevaluated first derivative with respect to lower case T of the second derivative with respect to U of the expression currently named E1. The second term is the 20th subexpression of a displayed expression currently named E2; this subexpression has been indicated with the light pen. The final term is the square of a subscripted function of X and Y. The notation may seem somewhat complex, but as will be seen, a complex notation is required to express in a compact way the many small steps required to solve a particular problem.

The infix operators are:

A←B	B is given name A. As such it is written on the disk. The value of ← is B.
¦A,N	The Nth subexpression of A is the value of ¦ Intensify the desired subexpression of A by pointing to its main connective with the light pen. Then type ¦A, and the computer will type N. If the expression has no main connective, point to one of its arguments and type ;¦A instead of ¦A. Consider all minus signs to be unary.
A=B	Equate A and B
A+B	A plus B
A-B	A minus B
A*B	A times B
A/B	A divided by B A/B*C is equivalent to A/(B*C)
A↑B	A to the power B

The functional, subscripting, and set notation is:

A(C,D,E)	A is a function with arguments C,D, and E.
A[I,J] (C,D,E)	$A_{I,J}$ is a function with arguments C,D, and E.
A[I,J]	$A_{I,J}$ is a variable.
(A,B,C)	This is a set with three elements. By convention (A) = A.

'	Either an expression or a variable name can be quoted. A function name always stands for itself. Quoting a function name means that its arguments will be evaluated but that the function will not be evaluated. For example let F(X,Y) = X-Y be a function and let X and Y be names for A; then F(X,Y) evaluates to 0 'F(X,Y) evaluates to F(A,A) F('X,Y) evaluates to X-A '(F(X,Y)) evaluates to F(X,Y). Quoting a function or variable name does not quote its subscripts. Numbers are taken as quoted automatically.
:	Causes the letters which follow it to be lower case for purposes of display.

As in CTSS, there are two editing characters:

?	Deletes all the characters of a command back to the initial ϕ.
"	Deletes only the immediately preceding character.

"ϕ" must be the first and last character of every command. ";" causes the current intensified subexpression to be raised one level. For example, if the A in A^B + C is intensified, then when ; is typed, A^B will be intensified.

Other available operators are:

ALLSUMEXPAND(EXP)	Applies SUMEXPAND to every summation in expression EXP.
BRINGOVER(EXP, X)	Subexpression X, which has been indicated with the light pen is brought to the other side of equation EXP.
COLLECT(EXP,SET)	Top level terms in EXP are collected on powers of the expressions in set SET.
DEPENDENCE(EXP)	Returns a set of the variable and function names in EXP.
DELSUBST(EXP,OLDDEL, NEWDEL)	$\frac{dx}{d\ \mathrm{OLDDEL}} \rightarrow \frac{dx}{d\ \mathrm{NEWDEL}}$ for each such subexpression in EXP.
DRV(X_1,N_1, ..., X_n,N_n, Y)	Differentiate Y N_i times with respect to X_i, for each i.
DRVDO(EXP,X)	All indicated derivatives with respect to X in EXP are carried out as far as possible.
DRVFACTOR(EXP, X, N)	$\frac{d^{N+M}f}{dx^{N+M}} \rightarrow \frac{d^M}{dx^M}\ (\frac{d^N f}{dx^N})$ for each such subexpression in EXP.
DRVZERO(EXP,X)	All derivatives with respect to X in EXP are set equal to zero.

EVALUATE(EXP,SET)	SET is a set of equations; whenever the left side of one of these equations can be matched to a subexpression in EXP, the right hand side is substituted. The left sides must be variables or functions. A match occurs whenever a binding of the function variables and subscripts can be made.
EXCHANGE(EXP)	If the top level connective of EXP is binary, its arguments are exchanged, right to left.
EXPAND(EXP)	Multiplies out all expressions of the form a*(b+c) in EXP. In addition, $\frac{d}{dx}(a+b) \rightarrow \frac{da}{dx} + \frac{db}{dx}$
FACTOROUT(EXP,FACTOR,Y)	The factor FACTOR is factored from each term of EXP. The third argument Y is optional. If Y is present, the factor FACTOR is renamed Y.
GROUP(SET)	The set SET of terms which have been indicated by the light pen in EXP are grouped within the associated sum or product. The value of GROUP is the grouped set of terms.
LEFT(EXP)	Returns the left argument of the main binary connective of EXP.
LIMIT(EXP,X,N)	Determines the limiting value of EXP as X approaches N.
MULTIPLYTHROUGH(EXP,X)	Multiplies each top level term of EXP by X.
NEWNAME()	Creates a name of the form Fn, where n is an integer.

NORMPOLY(EXP,X)	Every sum in EXP is treated as a polynomial in X and a power of X is factored out so that the lowest power of X in the polynomial will be zero.
REPLACE(E,X,Y)	Expression X replaces Y in the expression named E. Y is a term indicated with the light pen or a group of terms indicated with GROUP. If the light pen has been used to construct X, the resulting expression position is named HOLE. HOLE can then be used for the third argument. If X is equal to NIL, then the third argument is omitted from the expression named E.
RIGHT(EXP)	Returns the right argument of the main binary connective of EXP.
SIMPLIFY(EXP)	Simplifies expression EXP.
SOLVE(EXP,X)	Solves equation EXP for variable X as far as possible.
SPLIT(EXP)	Subparts of EXP are named and replaced by their names in EXP, so that EXP will contain less than 100 subexpressions.
SUBSTITUTE(EXP,X,Y)	Substitute X for each occurrence of Y in EXP.
SUMEACH(EXP)	$\Sigma(a+b) \rightarrow \Sigma a + \Sigma b$
SUMEXPAND(EXP)	Expands the finite summation EXP.
TERM(EXP,N)	Returns the Nth argument of the top level connective of EXP, or NIL if there is no Nth argument.
TRUNCATE(EXP,VAR,N)	Expands EXP up to power N in variable VAR.
SUM(I,N1,N2,Y)	Sum expression Y for values of I from N1 to N2.

ITG(X,L1,L2,Y)	Integrate Y with respect to X between limits L1 and L2.

Expressions which are assigned names are kept on the disk. The expression most recently computed always has the name LAST. When A←B is executed, if A is not "LAST" and is already the name of an expression, then this old value of A is given the name OLD. Thus, if A←A+2 is executed and then is found to be incorrect, the old value of A can be retrieved.

Operators used for input-output and disk storage are:

EDISPLAY(E)	Displays the expression named E on the PDP-6 scope.
EPRINT(E)	Prints out the internal form of the expression named E with PLS, PRD, EQN, and PWR in infix form; the other operators in prefix form.
EDELETE(E)	Deletes expression named E from the disk.

This completes the description of the PDP-6 commands.

The Poincaré-Lighthill Procedure Applied to $\ddot{x} + \omega^2 x = \epsilon x^3$

The Poincaré-Lighthill procedure is typical of a number of procedures used to find the first few terms in the asymptotic expansion of the function which is the solution to a mildly non-linear differential equation. The equation chosen here is that for a harmonic oscillator with a small forcing function. These solution procedures involve assuming a series expansion in powers of the small parameter ϵ for one or more of the parameters and variables in the equation, substituting these series into the differential equation, and thus obtaining a series of relations between the coefficients of like powers of ϵ. Each of these equations is then treated in turn by whatever methods seem appropriate. Thus it is in general necessary to see these equations before the next steps in the solution process can be determined.

When a typed command has been completed, the machine makes a response of acknowledgment. This standard response will be omitted in the dialogue to follow; only the typed commands and the displayed equations will be shown. A running discussion of the dialogue is included, and the displayed equations are shown as plotted by the CALCOMP plotter on the pages following their use. A photograph of the same equations displayed on the scope is shown at the end of the section. The reader should be aware that in the equation syntax used, more than one line of an expression can occur over a divide bar or within brackets. This is illustrated by equation Q16 in the last section.

Enter the differential equation.

```
#'E1←'(DRV(:T,2,X(:T)) + OMEGA↑2*X(:T) = EP*(X(:T))↑3)#
#EDISPLAY('E1)#
```

A new independent variable τ is introduced in order to stretch the time. Type in expressions for series expansions for X and t in terms of functions of τ. That is, a solution for X(τ) rather than for X(t) will be found. Since t depends on τ, equation E1 can be used to find an equation in derivatives of X(τ). As a final step the inverse relationship τ(t) will be found, so that X(τ) will give X(t).

```
#'E2←'(X(TAU) = SUM(I,0,INF,EP↑I*X[I](TAU)))#
#EDISPLAY('E2)#
#'E3←'(:T(TAU) = TAU + SUM(J,1,INF,EP↑J*:T[J](TAU)))#
#EDISPLAY('E3)#
```

In order to substitute dτ for dt it is necessary to apply the transformation

$$\frac{d^2x}{dt^2} \rightarrow \frac{d}{dt} \left(\frac{dx}{dt} \right)$$

to equation E1.

```
#'E4←DRVFACTOR(E1,':T,1)#
```

Display E4 for comparison with the substituted result below.

```
#EDISPLAY('E4)#
```

Now substitute dτ for dt, and X(τ) for X(t)

```
#'E5←SUBSTITUTE(DELSUBST(E4 ,'(DEL(:T)), '(DEL(TAU))/DRV('TAU,1,RIGHT(E3))),
                                          '(X(TAU)),'(X(:T)))#
#EDISPLAY('E5)#
```

Now substitute the series for X(τ) and perform the indicated differentiation with respect to τ.

```
#'E6←DRVDO(SUBSTITUTE(E5, RIGHT(E2),'(X(TAU))),'TAU)#
#EDISPLAY('E6)#
```

Now expand both sides to first order in ε.

```
#'E7←TRUNCATE(E6,'EP,1)#
#EDISPLAY('E7)#
```

$$\frac{d^2}{dt^2}X(t)+\omega^2\cdot X(t)=\epsilon\cdot X(t)^3 \quad \text{(E1)}$$

$$X(\tau)=\sum_{I=0}^{\infty}\epsilon^{I}\cdot X_{I}(\tau) \quad \text{(E2)}$$

$$t(\tau)=\tau+\sum_{J=1}^{\infty}\epsilon^{J}\cdot t_{J}(\tau) \quad \text{(E3)}$$

The zero order terms form the harmonic oscillator equation; the solution can be written down by inspection as Acos $\omega\tau$. Use the light pen to form an equation of the first order terms.

```
#'E8←!E7,6=;!E7,88#
#EDISPLAY('E8)#
```

Bring the terms in X_1 (τ) to the left side of the first order equation. Substitute for X_0 (τ) and carry out the indicated differentiation.

```
#'E9←SIMPLIFY(DRVDO(SUBSTITUTE(SOLVE(E8,'(X[1](TAU))),'(A*COS(OMEGA*TAU)),
                                                  (X[0](TAU))),'TAU))#
#EDISPLAY('E9)#
```

It is now necessary to substitute an identity for $\cos^3\omega\tau$ and to collect terms on $\sin\omega\tau$ and $\cos\omega\tau$.

```
#'E10←COLLECT(EXPAND(SUBSTITUTE(E9,'((COS(3*OMEGA*TAU)+3*COS(OMEGA*TAU))/4),
                 '((COS(OMEGA*TAU))↑3))),'((SIN(OMEGA*TAU),COS(OMEGA TAU))))#
#EDISPLAY('E10)#
```

Theoretical considerations require that the coefficients of $\cos\omega\tau$ and $\sin\omega\tau$ must be zero if there is to be a periodic solution for $X_1(\tau)$. From the coefficient of $\sin\omega\tau$ it is apparent that $t_1'(\tau)$ must be some constant C. This constant is determined from the coefficient of $\cos\omega\tau$.

```
#'E11←SIMPLIFY(SOLVE(SUBSTITUTE(!E10,44,'C,'(DRV(TAU,1,:T[1](TAU))))=0,'C))#
#EDISPLAY('E11)#
```

(E4) $$\frac{d}{dt}\left[\frac{d}{dt}x(t)\right]+\omega^2\cdot x(t)=\epsilon\cdot x(t)^2$$

(E5) $$\frac{\left\{\frac{d}{d\tau}\frac{\left[\frac{d}{d\tau}x(\tau)\right]}{\left\{\sum_{j=1}^{n}\left[\frac{d}{d\tau}t_j(\tau)\right]\cdot\epsilon^j+1\right\}}\right\}}{\left\{\sum_{j=1}^{n}\left[\frac{d}{d\tau}t_j(\tau)\right]\cdot\epsilon^j+1\right\}}+\omega^2\cdot x(\tau)=\epsilon\cdot x(\tau)^2$$

(E6) $$\frac{\left\{\frac{\left\{\sum_{i=0}^{n}\left[\frac{d^2}{d\tau^2}x_i(\tau)\right]\cdot\epsilon^i\right\}}{\left\{\sum_{j=1}^{n}\left[\frac{d}{d\tau}t_j(\tau)\right]\cdot\epsilon^j+1\right\}}+\frac{(-1)\cdot\left\{\sum_{j=1}^{n}\left[\frac{d^2}{d\tau^2}t_j(\tau)\right]\cdot\epsilon^j\right\}\cdot\sum_{i=0}^{n}\left[\frac{d}{d\tau}x_i(\tau)\right]\cdot\epsilon^i}{\left\{\sum_{j=1}^{n}\left[\frac{d}{d\tau}t_j(\tau)\right]\cdot\epsilon^j+1\right\}^2}\right\}}{\left\{\sum_{j=1}^{n}\left[\frac{d}{d\tau}t_j(\tau)\right]\cdot\epsilon^j+1\right\}}+\omega^2\cdot\sum_{i=0}^{n}\epsilon^i\cdot x_i(\tau)=\epsilon\cdot\left[\sum_{i=0}^{n}\epsilon^i\cdot x_i(\tau)\right]^2$$

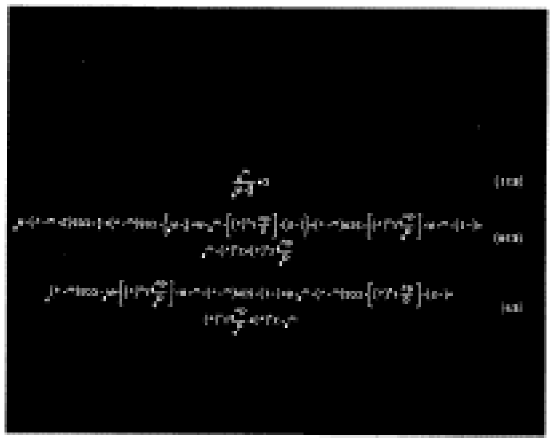
$$\text{(E7)}\quad \left\{(-2)\cdot\left[\frac{d^2}{d\tau^2}x_0(\tau)\right]\cdot\left[\frac{d}{d\tau}t_1(\tau)\right]+\frac{d^2}{d\tau^2}x_1(\tau)+(-1)\cdot\left[\frac{d^2}{d\tau^2}t_1(\tau)\right]\cdot\left[\frac{d}{d\tau}x_0(\tau)\right]+x_1(\tau)\cdot\omega^2\right\}\cdot\epsilon+\frac{d^2}{d\tau^2}x_0(\tau)+x_0(\tau)\cdot\omega^2=x_0(\tau)^2\cdot\epsilon$$

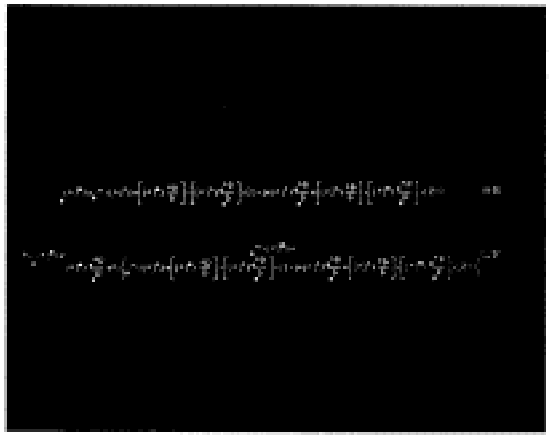
$$\text{(E8)}\quad (-2)\cdot\left[\frac{d^2}{d\tau^2}x_0(\tau)\right]\cdot\left[\frac{d}{d\tau}t_1(\tau)\right]+\frac{d^2}{d\tau^2}x_1(\tau)+(-1)\cdot\left[\frac{d^2}{d\tau^2}t_1(\tau)\right]\cdot\left[\frac{d}{d\tau}x_0(\tau)\right]+x_1(\tau)\cdot\omega^2=x_0(\tau)^2$$

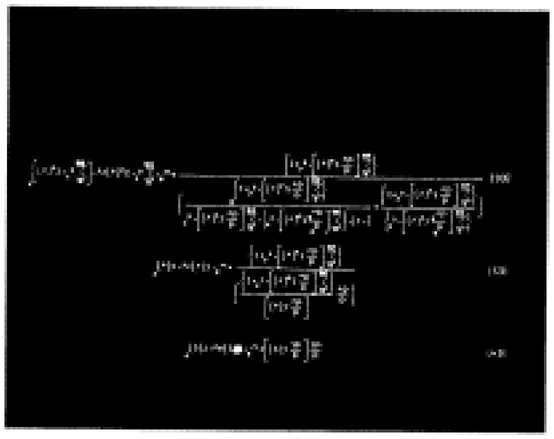
(E9)
$$\omega^2 \cdot X_1(\tau) + \frac{d^2}{d\tau^2} X_1(\tau)$$
$$= (-2) \cdot \left[\frac{d}{d\tau} t_1(\tau) \right] \cdot \mathrm{COS}(\omega \cdot \tau) \cdot \omega^2 \cdot A + (-1) \cdot \mathrm{SIN}(\omega \cdot \tau) \cdot \omega \cdot A \cdot \left[\frac{d^2}{d\tau^2} t_1(\tau) \right] + A^3 \cdot \mathrm{COS}(\omega \cdot \tau)^3$$

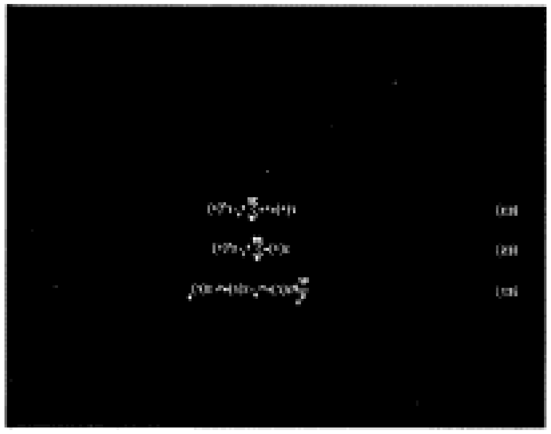
(E10)
$$\frac{d^2}{d\tau^2} X_1(\tau) + X_1(\tau) \cdot \omega^2$$
$$= (-1) \cdot \omega \cdot A \cdot \left[\frac{d^2}{d\tau^2} t_1(\tau) \right] \cdot \mathrm{SIN}(\omega \cdot \tau) + \left\{ (-2) \cdot \left[\frac{d}{d\tau} t_1(\tau) \right] \cdot \omega^2 \cdot A + \frac{3}{4} \cdot A^3 \right\} \cdot \mathrm{COS}(\omega \cdot \tau) + \frac{1}{4} \cdot \mathrm{COS}(3 \cdot \omega \cdot \tau) \cdot A^3$$

(E11)
$$C = \frac{\frac{3}{8} \cdot A^2}{\omega^2}$$


[illegible]


[illegible]


[illegible]


[illegible]

So $X_0 = A \cos\omega\tau$ and to first order $t = \tau + \epsilon \frac{3A^2}{8\omega^2}\tau$

Thus to first order $\tau \approx t(1 - \frac{\epsilon 3A^2}{8\omega^2})$ and $X_0 \simeq A \cos (1 - \frac{\epsilon 3A^2}{8\omega^2})t$

One effect of the non-linear term is thus seen to be an alteration in the frequency of the zero order term.

In conclusion, note the large number of small steps necessary to solve this problem. These are the result of doing almost the entire solution in the machine. In the case of a small problem such as this, some of these steps could be done by hand, the object here is to illustrate the steps which would be required for a larger problem. This problem also illustrates how rather lengthy intermediate calculations can lead to some rather concise results. The perturbation of the frequency in the zero order function is found to have a simple expression.

Plasma Accelerator Electrode Boundary Layers

The second problem is a duplication of the work in the first three sections of Chapter Three of a 1963 Masters Thesis by J. S. Draper for the M.I.T. Department of Aeronautics and Astronautics. This thesis investigates the laminar compressible boundary layer on the electrode walls of a direct-current crossed field plasma accelerator under very special physical conditions. Many of the assumptions used under these conditions are set forth in the paper "Electrode Boundary Layers in

Direct-Current Plasma Accelerators" by Jack L. Kerrebrock in the August 1961 Issue of the Journal of Aerospace Sciences. Kerrebrock's paper investigates a solution involving less mathematical manipulation than that undertaken by Masters Thesis student Draper.

In summary, the entire solution procedure is as follows;

1. Write down 5 non-linear partial differential equations:

   Momentum

   State

   Continuity

   Energy

   Electron mobility as a function of temperature.

These equations relate U stream velocity

V lateral velocity

t temperature

ρ density

μ electron mobility

P pressure

in terms of the independent variables x and y. The constants are:

j current

B magnetic field

C_p specific heat

K compressibility

G conductivity

R gas constant

2. The absence of variation in the y direction in the free stream is used to find the momentum and state equations there. These two reduced equations are solved for $\frac{dP}{dx}$ which is eliminated from the five main equations, since P is not a function of y.

3. The relation $H = C_p t + U^2/2$ is used to substitute derivatives of H for those of t in the energy equation which then becomes an enthalpy equation. t is thus eliminated from this equation. Simplification of the resulting expression requires introduction of the momentum relation. This step is performed because the enthalpy equation has a term proportional to $[1 - \frac{1}{P_r}]$ where P_r is $\frac{C_p\mu}{K}$ and can be approximated as 1, thus eliminating this term.

4. Next, a change of independent and dependent variables is made. The change of independent variables is such that it approximates a similarity transformation for low Mach number. These transformations change x and y to $\overline{x}$ and η. In addition $\frac{U}{U_\infty}$ is defined as f', $\frac{t}{t_\infty}$ as θ, and $\frac{H}{H_\infty}$ as g. The momentum and enthalpy equations are transformed, using the continuity and state equations as side conditions. $\overline{x}$ is then changed to M_∞. There result two non-linear differential equations in f and g and their derivatives with respect to η and M_∞.

5. f and g are then approximated as $f'(\eta,M_\infty) = b(M_\infty)\eta + c(M_\infty)\eta^2$, $g(\eta,M_\infty) = e(M_\infty)\eta + f(M_\infty)\eta^2$. These approximations are substituted into the two non-linear differential equations. The equations are then integrated with respect to η between the wall and the edge of the velocity boundary layer δu and the edge of the enthropy boundary

layer δe respectively. $b(M_\infty)$ and $c(M_\infty)$ are eliminated from the result by the relations $f'(\delta u,M_\infty) = 1$ and $g(\delta e,M_\infty) = 1$. There result two ordinary linear differential equations for the derivatives of δu, c, δe, and f with respect to M_∞.

6. Two more linear differential equations for δu, c, δe, and f are generated by choosing the coefficients of the approximations in step 5 so as to satisfy the momentum and enthalpy equations produced in step 4 exactly at the extremal points $f'' = 0$ and $g' = 0$.

7. The four resulting linear differential equations are solved for the derivatives $\frac{\partial \delta u}{\partial M_\infty}$, $\frac{\partial c}{\partial M_\infty}$, $\frac{\partial \delta e}{\partial M_\infty}$, and $\frac{\partial f}{\partial M_\infty}$ by Gaussian reduction. These four expressions are then numerically integrated with a Runge-Kutta method.

This problem has several interesting features. It is a demonstration of the notation use by workers in this area. The algebraic expressions are of a size difficult to manipulate by hand, but within the capabilities of current machines. The final symbolic result is large; it is difficult to write the corresponding numerical integration program correctly when this result must be input by hand, but here it is developed in the machine and could then be transformed into the required numerical program. Note that the symbolic steps are needed in order to cast the problem in terms of the independent and dependent variables of interest. The problem is characterized by the application of simplifying side conditions and physical assumptions. As such, it involves a number of manipulations for the purpose of expression condensation. This will be apparent from the following step by step reproduction of the first three sections of Chapter III. These steps bring the described solution through the application of the similarity transformation to the momentum equation.

Input the momentum equation:

```
#'D1←'(RHO*(U*DRV(X,1,U) + V*DRV(Y,1,U)) = DRV(Y,1,MU*
          DRV(Y,1,U)) - DRV(X,1,P) + :J*B)#
```

Input the energy equation:

```
#'D2←'(RHO*C[P]*(U*DRV(X,1,:T) + V*DRV(y,1,:T)) =
          DRV(Y,1,K*DRV(Y,1,:T)) + MU*(DRV(Y,1,U))↑2
          + U*DRV(X,1,P) + :J↑2/SIGMA#
```

The boundary layer solutions must match the free stream solution. The free stream values are indicated by the subscript ∞. At the free stream, there is no variation in the boundary layer with respect to y. To save rewriting, define sets containing the variables to be subscripted.

```
#'D3←'(RHO,SIGMA,U,:T,H)#
#'D4←'(RHO[INF],SIGMA[INF],U[INF],:T[INF],H[INF])#
```

Then in the free stream D1 and D2 become:

```
#'D5←SIMPLIFY(SUBSTITUTE(DRVZERO(D1,'Y),D4,D3))#
#'D6←SIMPLIFY(SUBSTITUTE(DRVZERO(D2,'Y),D4,D3))#
#EDISPLAY('D5)#
#EDISPLAY('D6)#
```

D5 and D6 will now be used to express some of the expressions in D1 and D2 in terms of the free stream quantities. In order to substitute for some of the terms in a sum, the terms must be grouped using the light pen. This is somewhat inconvenient.

```
#EDISPLAY(D1)#
```

$$(D5) \qquad RHO_{\infty}\cdot\left(\frac{d}{dX}U_{\infty}\right)\cdot U_{\infty}=-\frac{dP}{dX}+j\cdot B$$

$$(D6) \qquad RHO_{\infty}\cdot C_{p}\cdot\left(\frac{d}{dX}t_{\infty}\right)\cdot U_{\infty}=U_{\infty}\cdot\left(\frac{dP}{dX}\right)+\frac{j^{2}}{\sigma_{\infty}}$$

$$(D1) \quad RHO\cdot\left[U\cdot\left(\frac{dU}{dX}\right)+V\cdot\left(\frac{dU}{dY}\right)\right]=\frac{d}{dY}MU\cdot\left(\frac{dU}{dY}\right)-\frac{dP}{dX}+j\cdot B$$

Now the last two terms in D1 are replaced by the left side of D5.

```
#'D7+REPLACE ('D1,LEFT(D5), GROUP((!D1,33,!D1,38)))#
#EDISPLAY('D7)#
#'D8+LEFT(D2) = EXPAND(SUBSTITUTE(
        RIGHT(D2), RIGHT(SOLVE(D6,'(DRV(X,1,P)))), '(DRV(X,1,P))))#
#EDISPLAY('D8)#
```

The last two terms in D8 are now put in a factored form.

```
#'D8+REPLACE('D8,FACTOROUT(GROUP((!D8,62,!D8,78)),'(:J*2/SIGMA[INF]))
                              ' HOLE)#
#EDISPLAY('D8)#
```

In order to effect a cancellation, equation D8 will now be transformed by replacing the temperature, t, with the enthalpy, H, using the definition

$H = C_p t + U^2/2.$

First $H = C_p t + U^2/2$ is solved for temperature t.

```
#'D9+SOLVE('(H(X,Y) = C[P]*:T(X,Y) + U(X,Y)*2/2),'(:T(X,Y)))#
```

Now the substitution is made for t and t_∞ .

```
#'D10+EXPAND(SUBSTITUTE(D8,(DRV('X,1,RIGHT(D9)),SUBSTITUTE(DRV('X,1,RIGHT(D9)),
          D4,D3),DRV('Y,1,RIGHT(D9))), '( DRV(X,1,:T), DRV(X,1,:T[INF]),
          DRV(Y,1,:T))))#
```

$$\text{RHO}\cdot\left[U\cdot\left(\frac{dU}{dX}\right)+V\cdot\left(\frac{dU}{dY}\right)\right]=\frac{d}{dY}\text{MU}\cdot\left(\frac{dU}{dY}\right)+\text{RHO}_{\infty}\cdot\left(\frac{d}{dX}U_{\infty}\right)\cdot U_{\infty} \qquad \text{(D7)}$$

$$\text{RHO}\cdot C_{p}\cdot\left[U\cdot\left(\frac{dt}{dX}\right)+V\cdot\left(\frac{dt}{dY}\right)\right]=\frac{d}{dY}\left(\frac{dt}{dY}\right)\cdot K+\left(\frac{dU}{dY}\right)^{2}\cdot \text{MU}+\text{RHO}_{\infty}\cdot C_{p}\cdot\left(\frac{d}{dX}t_{\infty}\right)\cdot U-\frac{J^{2}\cdot U}{U_{\infty}\cdot\sigma_{\infty}}+\frac{J^{2}}{\sigma} \qquad \text{(D8)}$$

$$\text{RHO}\cdot C_{p}\cdot\left[U\cdot\left(\frac{dt}{dX}\right)+V\cdot\left(\frac{dt}{dY}\right)\right]=\frac{d}{dY}\left(\frac{dt}{dY}\right)\cdot K+\left(\frac{dU}{dY}\right)^{2}\cdot \text{MU}+\text{RHO}_{\infty}\cdot C_{p}\cdot\left(\frac{d}{dX}t_{\infty}\right)\cdot U+\frac{J^{2}\cdot\left(-\frac{U}{U_{\infty}}+\frac{\sigma_{\infty}}{\sigma}\right)}{\sigma_{\infty}} \qquad \text{(D8)}$$

Now, there are a number of tedious grouping steps.

```
#EDISPLAY('D10)#
```

First, factor RHO out of two of the terms on the left side of the equation.

```
#'D11←REPLACE('D10,FACTOROUT(GROUP((!D10,48,!D10,21)),'RHO),'HOLE)#
#EDISPLAY('D11)#
```

Now factor RHO·U from the other two terms and bring them to the right side.

```
#'D11←BRINGOVER('D11,FACTOROUT(GROUP((!D11,5,!D11,45)),'(RHO*U(X,Y))))#
```

A machine matching operation would be better for the next factoring step which must be done twice when light-pen pointing is used.

```
#EDISPLAY('D11)#
```

The quantity $K/C_p\mu$ is factored out of two of the terms and set equal to $1/P_r$.

```
#'D11←REPLACE('D11,FACTOROUT(!D11,34,'(K/(C[P]*MU)),'(1/P[R])),'HOLE)#
#EDISPLAY('D11)#
#'D11←REPLACE('D11,FACTOROUT(!D11,25,'(K/(C[P]*MU)),'(1/P[R])),'HOLE)#
```

$C_p\mu/K$ is the Prandtl number P_r. It is close to 1, and setting it to 1 will effect a simplification if the identity

$$\mu\left(\frac{dU}{dY}\right)^2 = \frac{d}{dY}[\mu \quad \cdot U(X,Y)\frac{d\ U(X,Y)}{dY}] - U \cdot \frac{d}{dY}[\mu\frac{dU}{dY}]$$

is also substitued. This simplification means that the heat conduction away from a point is just equal to the viscous dissipation at that point.

```
#'D11←SIMPLIFY(SUBSTITUTE(D11,'(1,DRV(Y,1,MU*U(X,Y)*DRV(Y,1,
U(X,Y))) -U(X,Y)*DRV(Y,1,MU*DRV(Y,1,U))),'(P[R],MU*(DRV(Y,1,U))↑2)))#
#EDISPLAY('D11)#
```

$$(D10)\ -U(X,Y)\cdot\left[\frac{d}{dY}U(X,Y)\right]\cdot U\cdot RHO+\left[\frac{d}{dY}H(X,Y)\right]\cdot U\cdot RHO-U(X,Y)\cdot\left[\frac{d}{dX}U(X,Y)\right]\cdot U\cdot RHO+\left[\frac{d}{dX}H(X,Y)\right]\cdot U\cdot RHO$$

$$=-\frac{d}{dY}\frac{K\cdot U(X,Y)\cdot\left[\frac{d}{dY}U(X,Y)\right]}{C_p}+\frac{d}{dY}\frac{K\cdot\left[\frac{d}{dY}H(X,Y)\right]}{C_p}+MU\cdot\left(\frac{dU}{dY}\right)^2+U\cdot\left[\frac{d}{dX}H_e(X,Y)\right]\cdot RHO_e$$

$$-U\cdot U_e(X,Y)\cdot\left[\frac{d}{dX}U_e(X,Y)\right]\cdot RHO_e+\frac{J^2}{\sigma}-\frac{U\cdot J^2}{U_e\cdot\sigma_e}$$

$$(011)\quad -U(X,Y)\cdot\left[\frac{d}{dY}U(X,Y)\right]\cdot U\cdot RHO+RHO\cdot\left\{\left[\frac{d}{dX}H(X,Y)\right]\cdot U+\left[\frac{d}{dY}H(X,Y)\right]\cdot V\right\}-U(X,Y)\cdot\left[\frac{d}{dX}U(X,Y)\right]\cdot U\cdot RHO$$

$$=-\frac{d}{dY}\frac{K\cdot U(X,Y)\cdot\left[\frac{d}{dY}U(X,Y)\right]}{C_p}+\frac{d}{dY}\frac{K\cdot\left[\frac{d}{dY}H(X,Y)\right]}{C_p}+MU\cdot\left(\frac{dU}{dY}\right)^2+U\cdot\left[\frac{d}{dX}H_e(X,Y)\right]\cdot RHO_e$$

$$-U\cdot U_e(X,Y)\cdot\left[\frac{d}{dX}U_e(X,Y)\right]\cdot RHO_e+\frac{j^2}{\sigma}-\frac{U\cdot j^2}{U_e\cdot\sigma_e}$$

(D11)

$$RHO\cdot\left\{\left[\frac{d}{dX}H(X,Y)\right]\cdot U+\left[\frac{d}{dY}H(X,Y)\right]\cdot V\right\}$$

$$= -\frac{d}{dY}\frac{K\cdot U(X,Y)\cdot\left[\frac{d}{dY}U(X,Y)\right]}{C_p}+\frac{d}{dY}\frac{K\cdot\left[\frac{d}{dY}H(X,Y)\right]}{C_p}+MU\cdot\left(\frac{dU}{dY}\right)^2+U\cdot\left[\frac{d}{dX}H_\infty(X,Y)\right]\cdot RHO_\infty$$

$$-U\cdot u_\alpha(X,Y)\cdot\left[\frac{d}{dX}u_\infty(X,Y)\right]\cdot RHO_\alpha+\frac{j^2}{\sigma}-\frac{U\cdot j^2}{U_\infty\cdot\sigma_\alpha}-RHO\cdot u(X,Y)\cdot\left\{-\left[\frac{d}{dY}U(X,Y)\right]\cdot V-\left[\frac{d}{dX}U(X,Y)\right]\cdot U\right\}$$

$$(D11)\qquad RHO\cdot\left\{\left[\frac{d}{dX}H(X,Y)\right]\cdot U+\left[\frac{d}{dY}H(X,Y)\right]\cdot U\right\}$$

$$= \quad -\frac{d}{dY}\frac{U(X,Y)\cdot\left[\frac{d}{dY}U(X,Y)\right]\cdot MU}{P_r}+\frac{d}{dY}\frac{K\cdot\left[\frac{d}{dY}H(X,Y)\right]}{C_p}+MU\cdot\left(\frac{dU}{dY}\right)^2+U\cdot\left[\frac{d}{dX}H_e(X,Y)\right]\cdot RHO_e$$

$$-U\cdot U_e(X,Y)\cdot\left[\frac{d}{dX}U_e(X,Y)\right]\cdot RHO_e+\frac{J^2}{\sigma}-\frac{U\cdot J^2}{U_e\cdot\sigma_e}-RHO\cdot U(X,Y)\cdot\left\{-\left[\frac{d}{dY}U(X,Y)\right]\cdot U-\left[\frac{d}{dX}U(X,Y)\right]\cdot U\right\}$$

Next, the substitution in D11 of the right side of equation D7 for its left side effects a nice cancellation.

```
#'D11←SIMPLIFY(REPLACE('D11,(-RIGHT(D7)),GROUP((!D11,117,!D11,122))))#
#'D11←LEFT(D11) = SIMPLIFY(EXPAND(SUBSTITUTE(RIGHT(D11),'(U,U[INF]),'(U,(X,Y),
                   U[INF](X,Y)))))#
#EDISPLAY('D11)#
```

The momentum equation D7 and the enthalpy equation D11 are now ready for the similarity transformation.

The transformations of independent variables are:

```
#'D12←'(XB(X) = ITG(X,0,X,P*U[INF]/(P[0]*U[0])))#
#'D13←' (ETA(X,Y) = (U[INF]/U[0])*((U[0]/(2*NU[0]*XB(X))↑(1/2))*
             ITG(Y,0,Y,RHO/RHO[0]))#
```

Next, to compute the required differentials.

First $$\frac{1}{dX} = \frac{d\eta}{dX} \cdot \frac{1}{d\eta} + \frac{d\bar{X}}{dX} \cdot \frac{1}{d\bar{X}}$$

```
#'D14←DRV('X,1,RIGHT(D13))*'(DEL(ETA))+DRV('X,1,RIGHT(D12))*'DEL(XB))#
#EDISPLAY('D14)#
```

D14 now contains the expression for η as a factor.

```
#'D14←REPLACE('D14,FACTOROUT(!D14,4,RIGHT(D13),'(ETA(X,Y))),'HOLE)#
#EDISPLAY('D14)#
```

η is now substituted for its definition and $(P \cdot U_\infty)/(P_0 \cdot U_0)$ is factored out.

```
#'D14←SIMPLIFY(FACTOROUT(SUBSTITUTE(D14,DRV('X,1,RIGHT(D12)),
            '(DRV(X,1,XB(X)))),'P*U[INF]/(P[0]*U[0])))#
#EDISPLAY('D14)#
```

(D11) $$\mathrm{RHO}\cdot\left\{\left[\frac{d}{dX}H(X,Y)\right]\cdot U+\left[\frac{d}{dY}H(X,Y)\right]\cdot U\right\}$$

$$=\frac{d}{dY}\left[\frac{d}{dY}H(X,Y)\right]\cdot \mathrm{MU}-U(X,Y)\cdot\left[\frac{d}{dY}\mathrm{MU}\cdot\left(\frac{dU}{dY}\right)\right]+U\cdot\left[\frac{d}{dX}H_e(X,Y)\right]\cdot \mathrm{RHO}_e-U\cdot U_e(X,Y)\cdot\left[\frac{d}{dX}U_e(X,Y)\right]\cdot \mathrm{RHO}_e$$

$$+\frac{j^2}{\sigma}-\frac{U\cdot j^2}{U_e\cdot\sigma_e}-\mathrm{RHO}\cdot U(X,Y)\cdot\left\{-\left[\frac{d}{dY}U(X,Y)\right]\cdot U-\left[\frac{d}{dX}U(X,Y)\right]\cdot U\right\}$$

(D11) $$\mathrm{RHO}\cdot\left\{\left[\frac{d}{dX}H(X,Y)\right]\cdot U+\left[\frac{d}{dY}H(X,Y)\right]\cdot U\right\}=\frac{d}{dY}\mathrm{MU}\cdot\left[\frac{d}{dY}H(X,Y)\right]+\mathrm{RHO}_e\cdot\left[\frac{d}{dX}H_e(X,Y)\right]\cdot U+\frac{j^2}{\sigma}-\frac{j^2\cdot U}{\sigma_e\cdot U_e}$$

$$(D14)\qquad -\frac{\left[\frac{d}{dX}XB(X)\right]\cdot NU_e\cdot 2\cdot U_e\cdot\left[\frac{U_e}{2\cdot NU_e\cdot XB(X)}\right]^{\frac{1}{2}-1}\cdot U_\omega\cdot\left(\int_0^Y\frac{RHO}{RHO_e}\,dY\right)\cdot DEL(ETA)}{\left[2\cdot NU_e\cdot XB(X)\right]^2\cdot 2\cdot U_e}+\frac{P\cdot U_\omega\cdot DEL(XB)}{P_e\cdot U_e}$$

$$(D14)\qquad -\frac{\frac{1}{2}\cdot ETA(X,Y)\cdot\left[\frac{d}{dX}XB(X)\right]\cdot DEL(ETA)}{XB(X)}+\frac{P\cdot U_\omega\cdot DEL(XB)}{P_e\cdot U_e}$$

The differential for 1/dY is

```
#'D15←DRV('Y,1,RIGHT(D13))*'(DEL(ETA))#
#EDISPLAY('D15)#
```

Expressions D14 and D15 are the required differentials for $\frac{d}{dX}$ and $\frac{d}{dY}$ respectively.

Now to transform the momentum equation D7. First, substitue for the differentials of the independent variables and the new normalized dependent variable f, defined by $U/U_\infty = f'(\bar{\bar{X}},\eta)$.

```
#'D16←SIMPLIFY(SUBSTITUTE(DELSUBST(DELSUBST(D7,'(DEL(X)),D14),
                         '(DEL(Y)),D15),'(U[INF](XB)*DRV(ETA,1,
                         :F(XB,ETA)),U[INF](XB),XB,ETA),'(U,U[INF],XB(X),
                         ETA(X,Y))))#
```

Now assume $\mu = \mu_0 t/t_0$.

```
#'D17←SIMPLIFY(SUBSTITUTE('D16,'(MU[0]*:T/:T[0],'MU))#
```

Use the equation of state to recognize that

$$\frac{d}{d\eta}\ (\rho T) = \frac{dP}{d\eta} = 0$$

since P is taken independent of Y. A substitution for P should therefore be made.

```
#EDISPLAY('D17)#
```

The machine responds with "EXPRESSION TOO LARGE"
Therefore, the left side of equation D17 will be treated first, the substitution for P will be deferred.

```
#'D18←LEFT(D17)#
```

$$\text{(D14)} \quad \frac{P \cdot U_{\infty} \left[\frac{\left(-\frac{1}{2}\right) \cdot ETA(X,Y) \cdot DEL(ETA)}{XB(X)} + DEL(XB) \right]}{P_0 \cdot U_0}$$

$$\text{(D15)} \quad \frac{RHO \cdot \left[\frac{U_0}{2 \cdot NU_0 \cdot XB(X)} \right]^{\frac{1}{2}} \cdot U_{\infty} \cdot DEL(ETA)}{RHO_0 \cdot U_0}$$

An expression for V will now be developed and substituted into D18. It was shown in Chapter II of the thesis that the continuity equation yields

$$V = -\frac{\rho_0}{\rho}\frac{1}{dX}\left(2U_0\nu_0\overline{X}\right)^{\frac{1}{2}} f(\overline{X},\eta)$$

Enter this expression.

```
#'D19←'(-(RHO[O]/RHO)*DRV(X,1,((2*U[O]*NU[O]*XB)↑(1/2)*:F(XB,ETA)))#
```

Now substitute the new independent variables:

```
#'D20←SUBSTITUTE(DELSUBST(D19,'(DEL(X)),D14),'(U[INF](XB),XB,ETA),
                '(U[INF],XB(X),ETA(X,Y)))#
```

Now, substituting this expression for V into the partially transformed left side of the momentum equation, D18, the entire expression is differentiated as far as possible, expanded and simplified.

```
#'D21←EXPAND(DRVDO(DRVDO(SUBSTITUTE(D18,D20,'V),'ETA),'XB))#
```

Now some shorthand will be introduced to make D21 easier to read.

```
#'D22←'(:F,U,F1,F2,F01,F11,U[INF])
#'D23←'(:F(XB,ETA),U(XB),DRV(ETA,1,:F),DRV(ETA,2,:F),DRV(XB,1,:F),DRV
              (ETA,1,XB,1,:F),U[INF](XB))#
#'D21←SUBSTITUTE(D21,D22,D23)#
#EDISPLAY('D21)#
```

It is now convenient to factor a large coefficient from each term and set it equal to 1 since it will also be factored from the other side and set to 1 there.

```
#'D24←SIMPLIFY(FACTOROUT(D21,'(RHO*U[INF]↑3*P/(2*P[O]*U[O]*XB)),1))#
#EDISPLAY('24)#
```

Returning to the right side of D17.

```
#'D25←RIGHT(D17)#
#EDISPLAY('D25)#
```

The deferred substitution for P is now made.

```
#'D27←REPLACE('D25,FACTOROUT(!D25,44,'((RHO*:T)/(RHO[0]*:T[0])),
          '(P/P[0])),'HOLE)#
```

The expression is now differentiated as far as possible, as was the left side.

```
#'D28←EXPAND(DRVDO(DRVDO(D27,'ETA),'XB))#
```

The simplifying notation is substituted.

```
#'D28←SUBSTITUTE(D28,D22,D23)#
#EDISPLAY('D28)#
```

Finally the large factor is removed from each term as it was from the right side.

```
#'D29←SIMPLIFY(FACTOROUT(D28,'(RHO*U[INF]↑3*P/(2*P[0]*U[0]*XB)),1))#
```

The two sides of the transformed momentum equation are now recombined.

```
#'D30←SIMPLIFY(D24-D29) = 0#
#EDISPLAY(D'30)#
```

Using some more relations developed earlier in the thesis, a final substitution will be made, from the equation of state at constant pressure.

$$(D21)\qquad -\frac{U_{\infty}^{3}\cdot F2\cdot RHO\cdot F01\cdot P}{U_{0}\cdot P_{0}}+\frac{\left(-\frac{1}{2}\right)\cdot U_{\infty}^{2}\cdot F2\cdot RHO\cdot f\cdot P}{XB\cdot U_{0}\cdot P_{0}}+\frac{F1^{2}\cdot\left(\frac{d}{dXB}U_{\infty}\right)\cdot P\cdot U_{\infty}^{4}\cdot RHO}{U_{0}\cdot P_{0}}+\frac{U_{\infty}^{3}\cdot F11\cdot P\cdot F1\cdot RHO}{U_{0}\cdot P_{0}}$$

$$(D24)\qquad -2\cdot F2\cdot F01\cdot XB-F2\cdot f+\frac{2\cdot F1^{2}\cdot\left(\frac{d}{dXB}U_{\infty}\right)\cdot XB}{U_{\infty}}+2\cdot F11\cdot F1\cdot XB$$

$$\mathrm{RHO}\cdot 2^{-\frac{1}{2}}\cdot U_e^{-\frac{1}{2}}\cdot NU_e^{-\frac{1}{2}}\cdot XB^{-\frac{1}{2}}\cdot U_\infty(XB)$$

$$\text{(D25)}\qquad \frac{-\left\{\dfrac{d}{dETA}\,\dfrac{MU_e\cdot t\cdot \mathrm{RHO}\cdot 2^{-\frac{1}{2}}\cdot U_e^{-\frac{1}{2}}\cdot NU_e^{-\frac{1}{2}}\cdot XB^{-\frac{1}{2}}\cdot U_\infty(XB)\cdot\left\{\dfrac{d}{dETA}U_\infty(XB)\cdot\left[\dfrac{d}{dETA}f(XB,ETA)\right]\right\}}{t_e\cdot \mathrm{RHO}_e}\right\}}{\mathrm{RHO}_e}$$

$$+\frac{\mathrm{RHO}_\infty\cdot P\cdot U_\infty(XB)^2\cdot\left\{\dfrac{\left(-\frac{1}{2}\right)\cdot ETA\cdot\left[\dfrac{d}{dETA}U_\infty(XB)\right]}{XB}+\dfrac{d}{dXB}U_\infty(XB)\right\}}{P_e\cdot U_e}$$

(D28) $$\frac{\frac{1}{2}\cdot P\cdot MU_e\cdot U_\infty^3\cdot\left(\frac{d^3f}{dETA^3}\right)\cdot RHO}{RHO_e\cdot P_e\cdot U_e\cdot NU_e\cdot XB}+\frac{\left(\frac{d}{dXB}U_\infty\right)\cdot U_\infty^2\cdot P\cdot RHO_\infty}{U_e\cdot P_e}$$

(D30) $$-2\cdot F2\cdot F01\cdot XB-F2\cdot f+\frac{2\cdot F1^2\cdot\left(\frac{d}{dXB}U_\infty\right)\cdot XB}{U_\infty}+2\cdot F11\cdot F1\cdot XB-\frac{2\cdot\left(\frac{d}{dXB}U_\infty\right)\cdot RHO_\infty\cdot XB}{U_\infty\cdot RHO}-\frac{MU_e\cdot\left(\frac{d^3f}{dETA^3}\right)}{RHO_e\cdot NU_e}=0$$

(D31) $$-2\cdot F2\cdot F01\cdot XB-F2\cdot f+\frac{2\cdot F1^2\cdot\left(\frac{d}{dXB}U_\infty\right)\cdot XB}{U_\infty}+2\cdot F11\cdot F1\cdot XB \qquad =0$$

$$-\frac{2\cdot\left\{\left[\frac{1}{2}\cdot(GAMMA-1)\cdot M_\infty^2-\theta_w+1\right]\cdot G+\theta_w+\left(-\frac{1}{2}\right)\cdot M_\infty^2\cdot(GAMMA-1)\cdot F1^2\right\}\cdot\left(\frac{d}{dXB}U_\infty\right)\cdot XB}{U_\infty}-\frac{MU_e\cdot\left(\frac{d^3f}{dETA^3}\right)}{RHO_e\cdot NU_e}$$

$\frac{\rho_\infty}{\rho} = \frac{t}{t_\infty}$ but $\frac{t}{t_\infty} = \theta$ so $\frac{\rho_\infty}{\rho} = \theta$, and from Chapter II,

$$\theta = (\theta_s - \theta_w)g + \theta_w - (\theta_s - 1)f^2$$

$$\theta_s = 1 + \frac{\gamma-1}{2} \cdot M_\infty^2$$

The final transformed equation is:

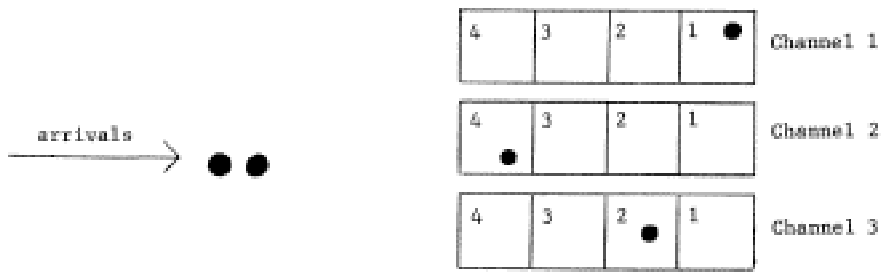
```
# 'D31←SIMPLIFY(SUBSTITUTE(REPLACE('D30,FACTOROUT('D30,38,'(RHO[INF]/RHO),
       '((THETA[S] - THETA[W])*G + THETA[W] - (THETA[S] - 1)*F1↑2)),'HOLE),
       '(1 + (GAMMA - 1)*M[INF]↑2/2),'(THETA[S]))$
# EDISPLAY('D31)$
```

In conclusion, since physical approximations are involved in the expression condensation, close man-machine interaction is required. The greatest draw backs to sufficient interaction are the input notation and the lack of sufficient facilities for abbreviation on output.

A Multiple Channel Queueing Problem

The third problem is taken from Interim Technical Report No. 13 of the M.I.T. Center for Operations Research entitled "A Class of Queueing Problems". This work is a 1955 Doctor's Thesis by H.N. Garber. The queueing situation shown by the example in Figure 1 is treated.

Figure 1



Arrival times at the queueing complex are exponentially distributed with mean λ. Arrivals enter any channel which becomes vacant and progress through several phases of service with exponentially distributed service times with mean $k\mu$. There are M channels and k phases of service in the general case. It is desired to find the probability distribution of the number of units in the system. The solution can be carried a number of steps for the general case. A set of equations relating the probabilities of the states are written, where a state of the system is taken as a certain number N in the system and a description of which phases of which channels are occupied. Each equation is then multiplied by the appropriate variables so that the summation of the equations will yield an equation with the generating function for the state probabilities as its left side. Unfortunately, the generating function also appears in a summation on the right side. The independent variables of the generating function are constrained to make each term of this summation equal to zero. This constraint is expressed in a change of independent variables. It is then shown that a summation over values of the new independent variables will yield the old generating function.

This generating function still involves the state probabilities for a partially full system as constants to be determined. There are a number of relations which can be used to determine these initial probabilities. As the third problem this evaluation will be carried out for the case of two channels with two phases of service.

The source of expression complexity is different in this problem. In the first problem complexity arose because a small parameter allowed an unsolvable problem to be split into a spectrum of solvable ones. In the second problem complexity was the result of the number of important phenomena in the physical process being described. Here, complexity is the result of the number of states in the process being described, all of which are very similar. In this problem there would be the best chance for reduction in complexity through proper notation.

This problem has been included so that the reader can better refine his intuitive notion of the types of mathematical operations and notation needed to solve problems in different areas of applied mathematics. Summation expansion, function evaluation, limits, and some more grouping operations are introduced.

The function in the transformed variables is:

```
#'Q1←'(K(Z,Q[1],Q[2]) = ((Z↑(2*:K + 1))*SUM(R,1,:K,(Z↑R)*
        A[R] * (:E↑(2*PI*I*Q[1]*R/:K)+:E↑(2*PI*I*Q[2]*R/:K))))/
        (2*(Z*W(Z↑:K) - ALPHA(Q[1],Q[2]))))#
```

Where W, α, and A_r are defined by:

```
#'Q2←'(W(Z↑:K) = 1 + THETA - THETA*Z↑:K)#
```

```
#'Q3←'(ALPHA(Q[1],Q[2]) = (:E↑(-2*PI*I*Q[1]/:K)+
        :E↑(-2*PI*I*Q[2]/:K))/2)#
#'Q4←'(A[R](Z) = P(1,R+1,0) - (2*W(Z↑:K) - 1)*P(1,R,0))#
#EDISPLAY('Q1)#
```

It is useful to have the denominator of Q1 contain only powers of Z^k. This is done by using the identity $(X-Y)\ (X^{k-1} \cdots Y^{k-1}) = (X^k - Y^k)$. At present the system contains no operators for achieving this goal, so it must be done by brute force.

```
#'Q5←LEFT(Q1) = '(SUM(J,0,:K-1,(Z*W(Z↑:K))↑J*ALPHA(Q[1],Q[2])↑(:K-1-J)))*
                SUBSTITUTE(RIGHT(Q1), '((Z↑:K)*(W(Z↑:K)↑:K) - ALPHA(Q[1],
                Q[2])↑:K), '(Z*W(Z↑:K) - ALPHA(Q[1],Q[2])))#
#EDISPLAY('Q5)#
```

Now substitute 2 for the number of phases of service, k. Then expand the summations.

```
#'Q6←SIMPLIFY(ALLSUMEXPAND(SIMPLIFY(SUBSTITUTE(Q5,2,':K))))#
#EDISPLAY('Q6)#
```

The substitution k = 2 is made in all the initial equations as well since they will be used several times.

```
#'Q1←SIMPLIFY(SUBSTITUTE(Q1,2,':K))#
#'Q2←SIMPLIFY(SUBSTITUTE(Q2,2,':K))#
#'Q3←SIMPLIFY(SUBSTITUTE(Q3,2,':K))#
#'Q4←SIMPLIFY(SUBSTITUTE(Q4,2,':K))#
```

Now write down an expression for the generating function as a summation of Q6 over the transformed independent variables, the q's.

(Q1) $$K(Z,Q_1,Q_2)=\frac{Z^{2\cdot k+1}\cdot\sum_{R=1}^{k}Z^{R}\cdot A_R\cdot\left(e^{\frac{2\cdot\pi\cdot I\cdot Q_1\cdot R}{k}}+e^{\frac{2\cdot\pi\cdot I\cdot Q_2\cdot R}{k}}\right)}{2\cdot\left[Z\cdot W\left(Z^{k}\right)-\mathrm{ALPHA}\left(Q_1,Q_2\right)\right]}$$

(Q2) $$W\left(Z^{k}\right)=\theta-\theta\cdot Z^{k}+1$$

$$(05)\qquad K(Z,\Theta_1,\Theta_2)=\frac{\frac{1}{2}\cdot Z^{2\cdot k+1}\cdot\left\{\sum_{j=0}^{k-1}\left[Z\cdot W\left(Z^k\right)\right]^j\cdot ALPHA\left(\Theta_1,\Theta_2\right)^{k-1-j}\right\}\cdot\sum_{p=1}^{k}Z^p\cdot A_p\cdot\left(e^{\frac{2\cdot\pi\cdot i\cdot\Theta_1\cdot p}{k}}+e^{\frac{2\cdot\pi\cdot i\cdot\Theta_2\cdot p}{k}}\right)}{\left[Z^k\cdot W\left(Z^k\right)^k-ALPHA\left(\Theta_1,\Theta_2\right)^k\right]}$$

$$(06)\qquad K(Z,\Theta_1,\Theta_2)=\frac{\frac{1}{2}\cdot\left[ALPHA\left(\Theta_1,\Theta_2\right)+Z\cdot W\left(Z^2\right)\right]\cdot Z^5\cdot\left[Z\cdot A_1\cdot\left(e^{\pi\cdot i\cdot\Theta_1}+e^{\pi\cdot i\cdot\Theta_2}\right)+Z^2\cdot A_2\cdot\left(e^{2\cdot\pi\cdot i\cdot\Theta_1}+e^{2\cdot\pi\cdot i\cdot\Theta_2}\right)\right]}{\left[Z^2\cdot W\left(Z^2\right)^2-ALPHA\left(\Theta_1,\Theta_2\right)^2\right]}$$

```
#Q7←'(H(Z↑2,V[1],V[2]) = (1/4)*SUM(Q[1],0,1,
        SUM(Q[2],0,1, ((V[1]/Z)*(-1)↑Q[1] + V[1]↑2/(Z↑2))*
        ((V[2]/Z)*(-1)↑Q[2] + V[2]↑2/(Z↑2))*K(Z,Q[1],Q[2]))))#
#EDISPLAY('Q7)#
```

Now expanding the summations, substituting the appropriate values of Q6, Q3, and k, and simplifying.

```
#'Q8←SIMPLIFY(EVALUATE(ALLSUMEXPAND(Q7),(Q6,Q3)))#
#EDISPLAY('Q8)#
```

To evaluate P(1,1,0) and P(1,2,0) certain known conditions are next imposed. Q8 is known to have a zero of order four in Z for all values of V_1 and V_2, so one would like to collect terms on Z^4. The simplest way to explore this would be to expand Q8 and collect terms on Z^4. Unfortunately, this leads to roughly a sixteen-fold growth in expression size and to memory overflow.

Inspection of Q8 shows that it might be rearranged while in factored form. As the first step, subexpressions which are polynomials in Z, V_1, or V_2, have factors of Z, V_1, or V_2 removed so that their lowest order term is of zero order in these.

```
#'Q81←SIMPLIFY(NORMPOLY(NORMPOLY(NORMPOLY(Q8,'Z),'(V[1])),'(V[2])))#
#EDISPLAY('Q81)#
```

Next the center two terms are combined.

```
#'Q82←REPLACE('Q81,FACTOROUT(GROUP((!Q81,85,!Q81,108)),
         '(A[2]/W(Z↑2))),'HOLE)#
#EDISPLAY('Q82)#
```

(a7) $$H\left(Z^2, U_1, U_2\right) = \frac{1\cdot\sum_{a_1=0}^{1}\sum_{a_2=0}^{1}\left[\frac{U_1\cdot(-1)^{a_1}}{Z}+\frac{U_1^2}{Z^2}\right]\cdot\left[\frac{U_2\cdot(-1)^{a_2}}{Z}+\frac{U_2^2}{Z^2}\right]\cdot K(Z, a_1, a_2)}{4}$$

(a8) $$H\left(Z^2, U_1, U_2\right)$$

$$=\frac{1}{4}\cdot\left\{\frac{\frac{1}{2}\cdot\left(\frac{U_1}{Z}+\frac{U_1^2}{Z^2}\right)\cdot\left(\frac{U_2}{Z}+\frac{U_2^2}{Z^2}\right)\cdot\left[Z\cdot W\left(Z^2\right)+1\right]\cdot Z^5\cdot\left(2\cdot Z\cdot A_1+2\cdot Z^2\cdot A_2\right)}{\left[Z^2\cdot W\left(Z^2\right)^2-1\right]}+\frac{\left(\frac{U_1}{Z}+\frac{U_1^2}{Z^2}\right)\cdot\left(-\frac{U_2}{Z}+\frac{U_2^2}{Z^2}\right)\cdot Z^8\cdot A_1}{W\left(Z^2\right)}\right.$$

$$\left.+\frac{\left(-\frac{U_1}{Z}+\frac{U_1^2}{Z^2}\right)\cdot\left(\frac{U_2}{Z}+\frac{U_2^2}{Z^2}\right)\cdot Z^8\cdot A_2}{W\left(Z^2\right)}+\frac{\frac{1}{2}\cdot\left(-\frac{U_1}{Z}+\frac{U_1^2}{Z^2}\right)\cdot\left(-\frac{U_2}{Z}+\frac{U_2^2}{Z^2}\right)\cdot\left[Z\cdot W\left(Z^2\right)-1\right]\cdot Z^5\cdot\left(-2\cdot Z\cdot A_1+2\cdot Z^2\cdot A_2\right)}{\left[Z^2\cdot W\left(Z^2\right)^2-1\right]}\right\}$$

The resulting term is then expanded.

```
#'Q83+REPLACE('Q82, EXPAND(!Q82,90), 'HOLE)#
#EDISPLAY('Q83)#
```

Now the other two terms are combined.

```
#'Q84+REPLACE('Q83,FACTOROUT(GROUP((!Q83,111,!Q83,30)),1/Q83!70),'HOLE)#
#EDISPLAY('Q84)#   EXPRESSION TOO LARGE
```

Q84 will not display, so it is reduced in size by naming a subpart.

```
#'Q85+SPLIT(Q84)#
#EDISPLAY('Q85)#
```

Next the numerator of the larger term in Q85 is arranged on powers of Z.

```
#'Q86+REPLACE('Q85,COLLECT(EXPAND(SUBSTITUTE(!Q85,31,F2,'F2)),'Z),'HOLE)#
#EDISPLAY('Q86)#   EXPRESSION TOO LARGE
#'Q87+SPLIT(Q86)#
#EDISPLAY('Q87)#
```

Forming a term from the renamed parts of Q87 one has for the coefficient of Z^2 in Q87.

```
#'Q88+SIMPLIFY(F2 + F3 + F4 + '(2*A1) + F5 + F6)#
#EDISPLAY('Q88)#
```

Looking at equation Q87, one can see that in a double power series expansion in V_1 and V_2, only the coefficient of $V_1^2V_2^2$ does not have a zero of order 4 at Z = 0. Setting this coefficient equal to zero one obtains by inspection of Q87:

$$\frac{A_2}{2W(Z^2)} + \frac{A_1 + Z^2 \cdot W(Z^2)A_2}{Z(Z^2 \cdot W^2(Z^2) - 1)} = 0$$

(Q81)

$$H\left(Z^2, U_1, U_2\right)$$

$$= \frac{1}{4} \cdot Z^2 \cdot U_1 \cdot U_2$$

$$\cdot \left\{ \frac{\frac{1}{2} \cdot (Z+U_1) \cdot (Z+U_2) \cdot \left[Z \cdot W\left(Z^2\right)+1\right] \cdot (2 \cdot A_1 + 2 \cdot Z \cdot A_2)}{\left[Z^2 \cdot W\left(Z^2\right)^2 - 1\right]} + \frac{(Z+U_1) \cdot (-Z+U_2) \cdot A_2}{W\left(Z^2\right)} + \frac{(-Z+U_1) \cdot (Z+U_2) \cdot A_2}{W\left(Z^2\right)} \right.$$

$$\left. + \frac{\frac{1}{2} \cdot (-Z+U_1) \cdot (-Z+U_2) \cdot \left[Z \cdot W\left(Z^2\right)-1\right] \cdot (-2 \cdot A_1 + 2 \cdot Z \cdot A_2)}{\left[Z^2 \cdot W\left(Z^2\right)^2 - 1\right]} \right\}$$

(082)

$$H\left(Z^2, U_1, U_2\right)$$

$$= \frac{1}{4} \cdot Z^2 \cdot U_1 \cdot U_2 \cdot \left\{ \frac{\frac{1}{2} \cdot (Z+U_1) \cdot (Z+U_2) \cdot \left[Z \cdot W\left(Z^2\right)+1\right] \cdot (2 \cdot A_1 + 2 \cdot Z \cdot A_2)}{\left[Z^2 \cdot W\left(Z^2\right)^2 - 1\right]} + \frac{A_2 \cdot \left[(Z+U_1) \cdot (-Z+U_2) + (-Z+U_1) \cdot (Z+U_2)\right]}{W\left(Z^2\right)} \right\}$$

$$+ \frac{\frac{1}{2} \cdot (-Z+U_1) \cdot (-Z+U_2) \cdot \left[Z \cdot W\left(Z^2\right)-1\right] \cdot (-2 \cdot A_1 + 2 \cdot Z \cdot A_2)}{\left[Z^2 \cdot W\left(Z^2\right)^2 - 1\right]}$$

(083) $$H\left(Z^2, U_1, U_2\right) = \frac{1}{4} \cdot Z^2 \cdot U_1 \cdot U_2 \cdot \left\{ \frac{\frac{1}{2} \cdot (Z+U_1) \cdot (Z+U_2) \cdot \left[Z \cdot W\left(Z^2\right) + 1\right] \cdot (2 \cdot A_1 + 2 \cdot Z \cdot A_2)}{\left[Z^2 \cdot W\left(Z^2\right)^2 - 1\right]} + \frac{A_2 \cdot \left(2 \cdot U_2 \cdot U_1 - 2 \cdot Z^2\right)}{W\left(Z^2\right)} \right\}$$

$$+ \frac{\frac{1}{2} \cdot (-Z+U_1) \cdot (-Z+U_2) \cdot \left[Z \cdot W\left(Z^2\right) - 1\right] \cdot (-2 \cdot A_1 + 2 \cdot Z \cdot A_2)}{\left[Z^2 \cdot W\left(Z^2\right)^2 - 1\right]}$$

(085) $$H\left(Z^2, U_1, U_2\right)$$

$$= \frac{1}{4} \cdot Z^2 \cdot U_1 \cdot U_2 \cdot \left\{ \frac{\left\{\frac{1}{2} \cdot (Z+U_1) \cdot (Z+U_2) \cdot \left[Z \cdot W\left(Z^2\right) + 1\right] \cdot (2 \cdot A_1 + 2 \cdot Z \cdot A_2) + F2\right\}}{\left[Z^2 \cdot W\left(Z^2\right)^2 - 1\right]} + \frac{A_2 \cdot \left(2 \cdot U_2 \cdot U_1 - 2 \cdot Z^2\right)}{W\left(Z^2\right)} \right\}$$

Enter the equation into the machine.

```
#'Q9←'(A[2]/(2*W(Z↑2)) + (A[1] + Z↑2*W(Z↑2)*A[2])/
        (2*(Z↑2*(W(Z↑2))↑2 -1)) = 0)#
```

Evaluating W, A_1 and A_2 one obtains at Z = 0

```
#'Q10←SIMPLIFY(SUBSTITUTE(EVALUATE(Q9,(Q4,Q2)),0,'Z))#
#EDISPLAY('Q10)#
```

Recognizing that by definition P(1,3,0) = P(1,1,0) this equation can be solved for P(1,2,0).

```
#'Q11←SIMPLIFY(SOLVE(SUBSTITUTE(Q10,'(P(1,0,0)),'(P(1,3,0))),'(P(1,2,0))))#
#EDISPLAY('Q11)#
```

Q11 can be simplified somewhat.

```
#'Q11←LEFT(Q11) = (-SIMPLIFY(FACTOROUT(!Q11,11,!Q11,29)/
        FACTOROUT(!Q11,43,!Q11,49)))#
#EDISPLAY('Q11)#
```

Let $u = z^2$, the generating function for the unconditional state probabilities $G(u) = \sum_{n=0}^{\infty} u^n p(n)$ can now be written. There are special case terms for n = 1, and n = 2. The other terms are found from H(u,1,1). The most compact formula for H is Q86.

```
#'Q12←'(G(U)) = '(P(0,0,0)) + 2*'U*('(P(1,1,0)) + RIGHT(Q11))
                 + SIMPLIFY(SUBSTITUTE(EVALUATE(RIGHT(Q86), (Q2, Q4)),
                            '(U↑(1/2), 1,1),'(Z,
                                   V[1],V[2])))#
```

The original transition equations give P(1,1,0) = θP(0,0,0), anticipating that P(1,3,0) may also occur, substitute P(1,3,0) = P(1,1,0) = θP(0,0,0).

(O87) $$H\left(Z^2, U_1, U_2\right)$$

$$=\frac{1}{4}\cdot Z^2\cdot U_1\cdot U_2\cdot\left\{\frac{\left[\left(F2+F3+F4+2\cdot A_1+F5+F6\right)\cdot Z^2+2\cdot W\left(Z^2\right)\cdot A_2\cdot Z^4+2\cdot A_1\cdot U_2\cdot U_1\right]}{\left[Z^2\cdot W\left(Z^2\right)^2-1\right]}+\frac{A_2\cdot\left(2\cdot U_2\cdot U_1-2\cdot Z^2\right)}{W\left(Z^2\right)}\right\}$$

(O88) $$2\cdot U_2\cdot A_2+2\cdot U_1\cdot A_2+2\cdot W\left(Z^2\right)\cdot U_2\cdot U_1\cdot A_2+2\cdot B_1+2\cdot U_2\cdot W\left(Z^2\right)\cdot A_1+2\cdot U_1\cdot W\left(Z^2\right)\cdot A_1$$

(O10) $$\frac{\frac{1}{2}\cdot\left[P(1,3,0)-(2\cdot\theta+1)\cdot P(1,2,0)\right]}{(\theta+1)}+\frac{1}{2}\cdot(2\cdot\theta+1)\cdot P(1,1,0)+\left(-\frac{1}{2}\right)\cdot P(1,2,0)=0$$

(O11) $$P(1,2,0)=-\frac{\left[\frac{1}{2}\cdot P(1,1,0)+P(1,1,0)\cdot\theta+\frac{\frac{1}{2}\cdot P(1,1,0)}{(\theta+1)}\right]}{\left[-\frac{1}{2}+\frac{\left(-\frac{1}{2}\right)}{(\theta+1)}-\frac{\theta}{(\theta+1)}\right]}$$

Then use Q11 to eliminate P(1,1,0).

```
#'Q13←SUBSTITUTE(Q12, RIGHT(Q11), '(P(1,2,0)))#
#'Q14←SIMPLIFY(SUBSTITUTE(Q13, '(THETA*P(0,0,0), THETA*P(0,0,0)),
                             '(P(1,3,0), P(1,1,0))))#
```

Q14 contains only the unknown P(0,0,0) which can be determined from the condition G(1) = 1.

```
#'Q15←SIMPLIFY(SUBSTITUTE(RIGHT(Q14),1,'U) = 1)#
```

The machine types out INDETERMINATE, indicating that $\infty * 0$ has been replaced by UNDEFINED.

```
#EDISPLAY('Q15)#
```

The operator LIMIT will be tried; this operator uses l'Hopital's Rule. It is slow, and so it should not be used when substitution will suffice.

```
#'Q16←LIMIT(RIGHT(Q14), 'U,1)#
#EDISPLAY('Q16)#
```

Q16 can be simplified by factoring and expansion.

```
#'Q17←FACTOROUT(EXPAND(Q16),'(1((3*THETA + 2)*(1-2*THETA))))#
#'Q17←LEFT(Q17)*EXPAND(RIGHT(Q17))#
#EDISPLAY('Q17)#
```

Q17 is equal to 1 and can be solved for P(0,0,0).

```
#'Q18←SOLVE (Q17=1, '(P(0,0,0)))#
#EDISPLAY('Q18)#
```

The expression for P(0,0,0) shows that the SOLVE routine could be improved. This expression is now substituted into Q14 in order to produce

(O11) $$P(1,2,0)=\frac{P(1,1,0)\cdot\left[\theta+2\cdot\theta\cdot(\theta+1)+2\right]}{(3\cdot\theta+2)}$$

(O15) $$P(0,0,0)+\frac{2\cdot P(0,0,0)\cdot\theta\cdot\left[\theta+2\cdot\theta\cdot(\theta+1)+2\right]}{(3\cdot\theta+2)}+2\cdot\theta\cdot P(0,0,0)+\text{UNDEFINED}\cdot\frac{1}{a}=0$$

the final expression for the generating function H.

```
#'Q19←SIMPLIFY(EVALUATE(Q14, Q18))#
```

Taking a census of Q19 shows that it is probably too large to display without being split.

```
#CENSUS(Q18)#
#EPRINT('LAST)#                1438
```

In the thesis this expression was evaluated numerically to form a table of values. This problem would provide a basis for further work in automatic simplification.

The preceding problem solutions show that the current system can be used for work on existing problems. No one part of the system is particularly weak, but there are many interesting possibilities for improvement.

(Q16)
$$P(0,0,0)+\frac{2\cdot\left[\theta+2\cdot(\theta+1)\cdot\theta+2\right]\cdot\theta\cdot P(0,0,0)}{(3\cdot\theta+2)}+2\cdot P(0,0,0)\cdot\theta$$

$$\frac{1}{4}\cdot\left\{\frac{16\cdot\left[\theta+2\cdot(\theta+1)\cdot\theta+2\right]\cdot P(0,0,0)\cdot\theta^{2}}{(3\cdot\theta+2)}+16\cdot P(0,0,0)\cdot\theta^{2}-\frac{4\cdot\left[\theta+2\cdot(\theta+1)\cdot\theta+2\right]\cdot\theta\cdot P(0,0,0)}{(3\cdot\theta+2)}\right\}$$

$$+\frac{+4\cdot P(0,0,0)\cdot\theta}{(-2\cdot\theta+1)}$$

(Q17)
$$\frac{\left[6\cdot\theta^{2}\cdot P(0,0,0)+7\cdot\theta\cdot P(0,0,0)+2\cdot P(0,0,0)+2\cdot\theta^{3}\cdot P(0,0,0)\right]}{(3\cdot\theta+2)\cdot(-2\cdot\theta+1)}$$

(Q18)
$$P(0,0,0)=\left[\frac{2\cdot\theta^{3}}{\left(-6\cdot\theta^{2}-\theta+2\right)}+\frac{2}{\left(-6\cdot\theta^{2}-\theta+2\right)}+\frac{7\cdot\theta}{\left(-6\cdot\theta^{2}-\theta+2\right)}+\frac{6\cdot\theta^{2}}{\left(-6\cdot\theta^{2}-\theta+2\right)}\right]^{-1}$$

Chapter III

ADDITIONAL GENERAL FEATURES

In the preceding chapter a system for manipulating mathematical expressions was introduced. In this chapter we discuss additonal features which would make this system more convenient in a wider variety of situations. Many of these would let the user tailor the program to his particular needs.

Whenever a user is provided with means for expanding and modifying a program, the question of efficiency becomes an important one. Efficiency normally results from planning a program function as a unit; building the function up bit by bit usually leads to waste. To an extent, though, it may be possible to design the system to guide the user, without unduely restricting him, into avoiding many serious inefficiencies. A careful analysis of the users' needs is required in order to determine what should be built into the program and what can be added later. Some of the features which the user might want are discussed in this chapter. They are divided into three classes: new mathematical structures, algorithms, and notation; bookkeeping; and "systems" instructions to the user and corrections of his mistakes.

Structures

There are many frequently-used mathematical structures; sets, rings, groups, fields, vector spaces, matrices, and tensors, to name a few. It might be possible to define some of these structures partly in terms of the others, but each structure will probably need its own unique data representation for really efficient computation. Some very good polynomial manipulation programs have been written (as noted in Chapter I); Maurer(4) uses a special representation for finite groups, and Hearn (5) has devised a data representation for tensors. Housekeeping programs should be written to accept the wide variety of data forms which can be expected. Special mathematical structures can form an important part of an input language even if they are not manipulated, since they let the mathematician state his problem in concise and familiar terms. For example, the initial steps in the solution of a set of equations can sometimes be expressed concisely in vector notation. The machine can perform the operations on the individual components.

Notation

Special mathematical structures will require symbols to represent them. The user should be able to define new symbols, and new arrangements of symbols. Hand-written input would make it easy to define new symbols; they could also be assigned to blank buttons or keys. In the current system, a string of input characters such as OMEGA can be assigned an output symbol such as ω. The system could be extended so that the user could define a new output symbol by choosing points to be displayed from a raster or by drawing a series of straight lines with the light pen. The user should also be able to make abbreviations such as x' for $\frac{dx}{dt}$. New combinations of symbols such as summation, $\sum_{i=0}^{\infty} x^i$, could be described in the picture language given later, but a more interesting possibility would be to generalize from hand drawn examples.

Algorithms

The user will want to add new mathematical functions and transformations in several ways. Special properties will be important for a given operation. For example, a simplification routine needs to know that $\sin^2 x + \cos^2 x = 1$, while a differentiation routine needs to know that $\frac{d \sin x}{dx} = \cos x$. A user may want to define a function only for certain arguments and later expand his definition. All the machinery of the "advice taker" programs, such as Teitleman's PILOT(1), could be used. When a transformation reaches arguments for which it is undefined, it can ask the user for advice. For example, in the current system the power series expansion routine will query the user about the poles of expressions, or simply ask for the expansion of a subexpression. It would be better if the user could input a general method for handling such expressions. However, in that case he must define the conditions under which it applies, and this requires a language which deals with concepts such as "rational function". The problem of combining new information with the old is an important one and should be pursued in the future.

New transformations could also be defined as combinations of old ones. If the string of user commands is remembered by the machine, the user could repeat an operation by refering back to his earlier commands. The string of commands could be displayed, so that the user could edit it or add comments. He might assign a name to a substring of commands and give it arguments to produce a function. These command strings could form a programming language with the addition of conditional expressions. Furthermore, the commands could be treated as mathematical expressions and should be legitimate data for the system, so that they could be simplified or otherwise transformed. The development of a command language will require the introduction of properties useful for describing mathematical expressions. For example, a method might apply only to expressions rational in sines and cosines. A specification of this method would require a check to see if the expression had this form. Such a language would allow a mathematician to describe his problem solution to the machine in the most general way, thereby increasing the number of problems which could be solved with the same string of commands. In applying the same commands to other problems, the machine could remember certain characteristics of the original problem and ask the user for help when new problems require further generalization or new methods.

Numerical Algorithms and Notation

A mathematical laboratory also needs a strong numerical capability. Even when symbolic manipulation is an essential part, most practical (and probably most theoretical) problems will require numerical methods somewhere in their solutions. Sometimes the equations describing a numerical method must be tailored to suit each problem. For example, an iterative technique may have a starting procedure which depends on the boundary conditions. This tailoring might be done with symbolic routines. In other cases, the equations to be processed numerically must be transformed symbolically before the numerical routines can be applied. This was the case in the second problem solved in the previous chapter.

Numerical evaluation can also be used in order to gain insight regarding exact or approximate symbolic solutions. For numerical work, mathematical structures must be represented as numerical arrays. Methods are needed for converting expressions from symbolic form to numerical form and the reverse. In addition, a method is needed to link with the large store of existing numerical routines. Finally, there must be input-output for numerical data in the form of tables and graphs. The MAP system and the Culler system provide many of the features needed, such as automatic scaling of graphs, formatting of tables, and the use of the light pen and parameter knobs for describing and altering numerical functions and constants.

Bookkeeping

Routine bookkeeping is often required for the solution of a large problem. The machine should assume this function in an inobtrusive manner. For example, the machine might keep a record of what expressions had been combined to form other expressions and display this information in the form of a graph. Touching some part of the graph with the light pen would cause the machine to display that part in greater detail. There might be a way for the user to name pages of expressions or to ask for all of the expressions with a certain property.

The machine could find missing relations needed to solve a set of equations, so that the user would not have to specify the arguments completely.

There would probably be many questions which the machine could answer about past inputs. Of course, the machine can also collect data of interest to system designers, such as the distribution of properties of expressions to which a given transformation is applied.

User Errors

Finally, the machine must help the user overcome his difficulties in using the system. It might be possible to develop a systematic method of rejecting inputs which are syntatically or semantically meaningless. The formal structure of mathematics makes this easier than it would be for some other real-time systems. The machine can use heuristics to determine what the user might have meant. Optional instructive comments like those in the MAP system would also be useful.

As can be seen from this chapter, the working mathematical laboratory will be a very complicated system. Development of the means necessary in order to create such a system will be one of the important steps in the next generation of programming research.

Chapter IV

INTRODUCTION TO THE REMAINING CHAPTERS

The remaining chapters describe the system in great detail and suggest possibilities for further work. These chapters are largely independent of one another and can be read in any order. Each chapter or section becomes more detailed toward the end. Therefore, one strategy would be to read the first part of each chapter and then read some of them to the end. Chapter VII, Chapter IX, and the first part of Chapter VI contain the most significant results.

Chapter V

PROBLEM AREAS

In this short chapter we mention a number of classical mathematical transformations for which it should be possible to write computer programs. Some have already been investigated by others, but all of them need more study. For some areas it seems possible to find an algorithm which will handle many of the expressions which arise in practice. Examples of these are differentiation, solution of equations by Gaussian reduction, and certain matrix and tensor operations. Even though an algorithm may in theory handle all expressions of a given class, it may in fact be limited by space or time considerations. Often, however, it can be improved through use of the mathematical properties of the transformation to be performed. For example, a sum with a positive integer exponent can always be expanded by repeated multiplication followed by collection of like terms. However, expansion of $(A_0 \cdots A_n)^K$ in this manner leads to $(n+1)^K$ terms, which combine to yield $\binom{n+K}{n}$ different terms. The $\binom{n+K}{n}$ terms can be found directly by the multinomial theorem; thus, in general, reducing the time and the maximum storage required. In the case of Gaussian reduction, savings can be made <u>on the average</u> through manipulation of zeros.

In other problem areas, algorithms can be used to transform only some examples, but the range of examples which can be transformed can probably be considerably extended with heuristic methods. Examples of these areas are limits, Laplace transforms, expansion in a Laurent series, integration of elementary functions in closed form, and solution of some differential equations.

Finally, in the case of non-linear differential equations, for example, it seems that only isolated special cases could be solved by the machine. It is, however, difficult to estimate á priori whether further research will lead to useful methods. As stated in the introduction, integration of the elementary functions in closed form has been proved recursively unsolvable. Such a proof not only affirms the futility of finding a general scheme but can also indicate which subclasses may be impossible. Perhaps more such proofs can be found. In the absence of such a proof, another measure of problem difficulty is the extent to which the solution is a global function of the input mathematical expressions making up the problem statement. When the solution is not a global function, then local properties of the input may give clues to local properties of the solution, thus making it possible to guess at the form of the answer.

Mathematical transformations should be a good medium for the investigation of computer problem solving.

Chapter VI

INTERNAL REPRESENTATION AND MANIPULATION OF MATHEMATICAL EXPRESSIONS

Introduction

The main body of this long chapter contains detailed descriptions of the effects of the LISP routines which have been written to manipulate mathematical expressions. This detailed description is introduced by a general discussion of the problems and possibilities involved in writing such routines. As the structures to be manipulated become more complex, the methods for representing them and for describing their transformation become, in the current usage, more a matter of choice. This is because the complex structures are described with a very general syntax, the ambiguities being resolved through context, and because there are fewer well established conventions for representing these structures. Therefore, the problem of manipulating the elementary functions will be discussed in some detail before more general structures involving sets are introduced.

The Elementary Functions

A. Introduction

The expressions for the elementary functions are defined recursively as follows. All floating point numbers, integers, and pairs of integers representing rational numbers are elementary functions. If u and v are elementary functions, then $u+v$, $u \cdot v$, u^v, u/v, $\log u$, and $-u$ are elementary functions. The trigonometric functions may be represented explicitly. In addition, arbitrary functions will be allowed. These functions are assumed only to be differentiable. It is allowed to write $a+b+c$ and $a \cdot b \cdot c$; the program assumes association.

Normally, transformation of these expressions involves two steps. Certain properties of the source expression are investigated and then the results of this investigation control the flow of control as the output expression is generated, often by combining parts of the source expression. It is difficult to list the important properties of expressions without examining actual problems. Some useful properties are those used by Slagle's Integration program SAINT and the routines described at the end of this chapter.

B. Properties of Elementary Functions

The properties can be classified according to their complexity as follows. Those from SAINT are indicated by (S).

1. Tests involving no investigation of subparts of the expression.
   a. Is the expression an integer. (S)
   b. Is the expression the variable of integration. (S)
   c. Expression is 0.
   d. Expression is 1.

2. Tests which are applied to the expression and its arguments in a completely recursive manner; that is, the test which is applied to the expression involves applying the identical test to the arguments of the expression, unless a terminating condition is met.
   a. Depth: the maximum number of arguments within arguments. (S)
   b. Length: the number of subexpressions in the expression. (S)
   c. Algebraic function: the expression contains only the algebraic operators and constant exponents. (S)
   d. Dependent on a given variable. (S)
3. Tests which involve applying tests to the arguments of an expression which are a function of the main connective of the expression.
   a. Rational function: if the main connective is POWER, then the exponent must not be a rational function, but rather an integer. (S)
   b. Rational function of sines and cosines: if the main connective is a sine or cosine, then the argument must be the variable of integration. (S)
   c. Exponential: as in b., the expression contains a subexpression of the form c^v. (S)
   d. Integral with indefinite limits.
   e. Integral with respect to a given variable.
   f. Derivative operator applied to an arbitrary function.
   g. Lowest power of a variable in power series expansion of an expression.

h. SUM with numerical lower limit.

i. SUM with numerical or infinite upper limit.

4. Tests which involve the comparison of certain subparts of the expression.

   a. Exponent base: there is some constant, b, and variable, x, such that any subexpression which is an exponential is of the form b^{nx}, where n is an integer. (S)

   b. Comparison of arguments of a product or sum for equality.

   c. All terms of a sum contain a common factor.

5. Non-recursive structures made up of parts each of which has certain properties.

   a. The expression contains a non-constant sum raised to an integer power. (S)

   b. The expression contains a non-constant sum as a factor of a product. (S)

   c. There is a subexpression of the form $c_3 + c_2v + c_1v^2$; $c_1 \neq 0$, $c_2 \neq 0$. (S)

   d. The expression is the product of two factors with certain properties. (S)

   e. The expression is the product of a factor with given properties and the expression $\cos^{2n+1}v$, where v is a variable. (S)

   f. The expression is an exponential with constant exponent.

g. Is expression of the form x*n; n an integer.

6. Some transformations require arbitrarily complex computations on the parts of the expression in order to establish the flow of control.

    a. Comparison of ranges of several summations and grouping together of like segments of the ranges.

    b. Generations of the lower order terms of a multinomial expansion.

C. <u>Methods for Testing Properties of Elementary Functions</u>

There are several mechanisms which can be used to determine if a given expression lies in one of these classes. The best method depends on which of the six named classes is involved as well as the associated constructive operations and the nature of the expressions under consideration.

Many transformations can be applied in turn to larger and larger subparts of the expression. Others are applied to certain parts of the expression as a whole. While those of the most complex type must make use of intermediate results as well as the initial expression.

It would be unlikely that any one scheme would be the most efficient for every combination of analysis and construction. The various transformations should be handled by methods appropriate to their complexity.

The efficiency of a transformation must be taken as an average over the set of expressions to which it is applied. Very often a transformation is applicable to only some of the expressions to which it is applied. Since a great deal of time can be spent in attempting to apply irrelevant transformations, the efficiency of a transformation might be evaluated in two ways: the average time required for application to a random expression, and the average time required for application to an expression to which it is appropriate. As expressions are transformed, more information about them will become known; it might be possible to pass this information along so as to improve the efficiency of further transformations. Furthermore, the physical situation which the expressions represent may give clues to their properties. Preserving such information involves trading space for time.

The expressions being considered form a tree structure; the main connective is considered to be at the top, with the branches hanging down. Some methods typically come down from the top, others come up from the bottom; some first go one way and then the other. Class 1 tests are applied at the top.

Class 2 tests can be applied going either way. For instance, in the case of depth, one can come down the tree incrementing the depth count at each node and then take the maximum of the counts at the ends of the branches, or one can come up the tree, incrementing the maximum branch count at each node. The number of incrementing and binary maximum operations is the same in either case.

The situation is more interesting for class 3 tests. Since the test to be applied to some of the subexpressions depends on the main connective above, all possible tests must be applied if one comes up from the bottom. Coming down from the top one can bring the information about the main connective along and apply only the needed test. If several tests are applied at once, then most of the tests will probably be needed on a given subexpression and not so much is lost.

An advantage of applying all tests to a subexpression is that identical subexpressions can be treated in the same manner since they are not distinguished by context. If several tests can be computed at once by bit manipulations,then the extra cost would be less. It seems that the great loss is probably that many of the tests, if applied from the top, would fail or succeed before all of the subexpressions had been tested, so that a saving would result from some subexpressions not being tested at all. (This seems to be a weakness of K. Korsvold's program).

The tests in class 4 involve the comparison of expressions for equivalence. Because of the defining relations, equivalent expressions can be written in several forms. For instance, the factors of sums and products can be written in any order. Some existing schemes impose some additional constraints on the way expressions are written so that there will be a cannonical form. Another possibility is to compute a hash code number for each expression which is invariant under the alternate ways of writing the expression. The method of doing this and the merits of the scheme are discussed in the next chapter.

The exponent base operation is the first test which requires comparison of subexpressions located at arbitrary nodes of the expression tree. A program which goes up or down the expression tree must have some way to communicate the information at one node to itself when it is processing another node.

The tests in class 6 are those which do not fit into one of the other classes. Their existence indicates that although programs which move up and down the expression tree are useful in a large number of cases, there will be tests for which this type of executive structure is very inefficient. Programs which carry out complex algorithms involving many forms of intermediate results will be needed. To introduce concepts which will be useful in such programs, operations for constructing elementary functions will now be considered.

D. Constructing Elementary Functions

All constructions which form elementary functions from other elementary functions must do so either by substitution or by combining the input functions with the elementary operators. Useful intermediate forms are a string of terms which are to be factors in a sum or product. Keeping such a string removes the need to test each time before adding new terms.

E. Specification of Algorithms for Manipulating Elementary Functions

No matter what level or flexibility a language for algorithm specification is to have, to be convenient and efficient it must reflect the structure of the algorithms to be specified. Based on the preceding discussion the following types of operations are needed. One begins with a basic program sequence that goes down and up the expression tree. When this sequence is associated with a specific set of rules, it is given a name by which it can be called. There are rules for the downward pass and the upward pass. The rules for the downward pass consist of a conditional specification of what is to be applied to the arguments, and a method of reserving information for lower levels and also for any following operation. The conditional is a match containing the operator and functions or exact match on the arguments. The arguments and groups of them can be given mnemonic names to be used in this match. There is a way to name all arguments satisfying some match. Matched arguments can be given special names to be used in the rest of the specifications. There is a way to break out immediately with the answer on either the upward or the downward pass. The upward pass consists of another match and a construct. The construct employs a pattern but the operators in the pattern which have variable arguments are simplified if possible. The construct pattern can contain functions. FLIP(III-1) might be useful in such a language

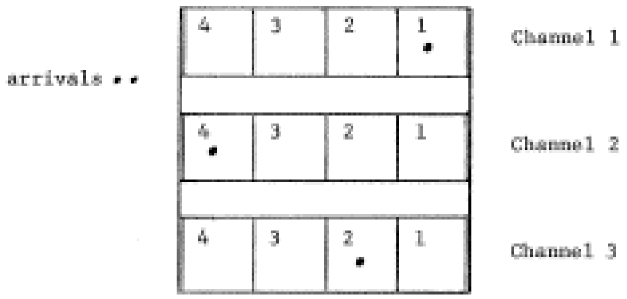
Representation of Sets

A set might be represented by giving its name and some of its properties, or by giving a general member and directions on how to specialize this general member to any particular member, or by giving all of the members. Members of ordered sets can be indexed to form arrays. More than one of these representations could be used simultaneously. Typical sets are an indexed summation of terms, or an indexed set of equations. In collecting terms one might want to combine several summations of the form $\sum_{i=m}^{n} a_i x^i$ to form a new set of summations, no two of which contain the same power of x. This amounts to combining members of different sets; that is, the coefficients, which members have the same value of a property, that is power of x. The summation index makes it possible to carry out this operation without explicitly testing each member of each set. This is a very general form in which many algorithms dealing with sets can be expressed, but greater computer efficiency and clarity of expression might result from using special concepts.

As another example of the use of a set concept, consider the placement of the coefficients of a polynomial into an array, so that the Routh test can be applied. A series of terms is generated by combining these coefficients. The generation of these terms is best specified as operations on an array of expressions. The requirement that the resulting terms all have the same sign then leads to relations between their parameters.

Two further examples are similar. Any particular term in a multinomial expansion can be written down directly in terms of the base and the exponent. This is done by forming all combinations of powers of the variables which fulfill certain restrictions. Computation with integers is used to determine the flow of control as the powers are formed. A similar but more complex case is the formation of terms in the power series of the multidimensional transformation of a non-linear differential equation as described by Van Trees.(V2)

As a final example, consider the multiple channel, multiple phase queueing system described by Garber and discussed in Chapter II. The basic equations describe a system such as the one shown below:



A system of m channels, each with k phases of service, is described by the total number n of units in the system and the state r_j of each channel. The arrival rate is λ and the service rate is μ. The equations are

$$(n+m\theta)\ p\ (n;r_1,\ r_2,\ \cdots\ r_n,\ 0,\ \cdots\ 0)$$

$$= \sum_{j=1}^{n}\ p(n;r_1 + \delta_{j,1},\ r_2 + \delta_{j,2},\ \cdots\ r_n + \delta_{j,n},\ 0,\ 0\ \cdots\ 0)$$

$$+ \sum_{j=n+1}^{m}\ p(n+1;r_1,r_2,\ \cdots,\ r_n,\ \delta_{j,n+1},\ \delta_{j,n+2},\ \cdots,\ \delta_{j,m})$$

whenever $n \leq m$ and subject to:

$$k\mu\ p(n;r_1,\ \cdots,\ r_{i-1},\ k+1,\ r_{i+1},\ \cdots,\ r_n,\ 0,\ \cdots\ 0)$$

$$= \frac{\lambda}{m-n+1}\ p(n-1;r_1,\ \cdots,\ r_i,\ 0,\ r_{i+1},\ \cdots,\ r_n,\ 0,\ 0,\ \cdots,\ 0)$$

$$+ \delta_{m,n}\ k\mu p\ (m+1;r_1,\ \cdots,\ r_{i-1},\ 1,\ r_{i+1},\ \cdots,\ r_m)$$

whenever $r_i = k$. While if $n > m$

$$(1+\theta)p\ (n;r_1,\ r_2,\ \cdots,\ r_m) = \frac{1}{m}\ \sum_{j=1}^{m}\ p(n;r_1 + \delta_{j,1},\ r_2 + \delta_{j,2},\ \cdots,\ r_m + \delta_{j,m})$$

$$+ \theta\ p\ (n-1;\ r_1,\ r_2,\ \cdots,\ r_m)$$

subject to

$$p(n;r_1,\ \cdots,\ r_{i-1},\ k+1,\ r_{i+1},\ \cdots,\ r_m) \equiv p(n+1;\ r_1,\ \cdots,\ r_{i-1},\ 1,\ r_{i+1},\ \cdots,\ r_m)$$

when $r_i=k$. It is understood that although only those equations for which the full channels have the lowest index have been written, there is in fact an equation for each permutation.

Note that this representation involves the use of several external conditionals, some written in English, as well as the internal conditional δ_{ij}. In addition, it contains the inductive "..." notation and the English statement about permutations.

Some of the external English conditional statements arise because the δ_{ij} does not provide enough generality as an internal conditional. The "..." notation arises in part because there is no standard way to mention a particular subscript of a quantity which is indexed by a number of subscripts which is itself a variable. It is for this same reason that the English statement about the permutations is made. The syntax used has great flexibility, but does not contain concepts appropriate to this statement. If the expression is represented internally in the computer in this form, it will be difficult to write algorithms to transform it.

An alternate form of notation would involve introducing a general internal conditional, functional arguments, and some specific concepts about sets. The function SETSUM [SET, FUNCTION] has as value the sum of the quantities produced by applying the function FUNCTION to each element of set SET. The function SETSUBST [X, Y, Z] substitutes element X for element Y in set Z. The concept of bound variables is also used. The current element to which SETSUM is applying FUNCTION is bound to the name, element. In this manner, the expression can be written:

```
EQN (M,N,K,RSET,$\theta$) $\equiv$
  If N $\leq$ M then (N+M$\theta$) P[N,RSET] = SETSUM (RSET,
       if element = 0 then P[N+1, SETSUBST (1+element, element, RSET)]
    or if element = k then $\frac{\lambda}{M-N+1}$ P[N-1, SETSUBST (0, element, RSET)]
           + if M=N then $k_\mu$ P[M+1, SETSUBST (1, element, RSET)]
             else 0
    else P[N, SETSUBST (element + 1, element, RSET)]
  else (1+$\theta$) P[N,RSET] = SETSUM (RSET,
       if element = k then $\frac{1}{m}$ P[N+1, SETSUBST (1, element, RSET)]
                           + $\theta$ P[N-1, RSET]
       else $\frac{1}{m}$ P[N, SETSUBST (element + 1, element, RSET) + $\theta$P[N-1, RSET]
```

There are several alternative ways in which this expression could be written. The choice would depend largely on the operations to be performed. The typical procedure is to apply a number of steps and then to generate a special case from the general expression.

In closing, another area where the problem solution goes from the general to the particular is the area of vector spaces and matrices.

Methods of Specification

The amount of information about the input to be saved in the internal representation depends on the range of tasks which the machine is to perform in the man-machine interaction. One possible task is the arrangement of statements input in any order into an order which allows their sequential evaluation. In some cases missing steps or limits could be pointed out. This requires some representation of what statements belong together as well as properties of the statements such as the information needed to evaluate them. With respect to evaluation, the more problem oriented, rather than procedure oriented, the man-machine discourse becomes, the more necessary it is to represent objects by a set of properties which would allow generation of the object by an appropriate routine. This is true, for example, of sets representing permutations and combinations subject to certain restrictions.

Such properties are also needed if the machine is to perform difficult problem solving tasks. Often, for example, the factors of a polynomial can be found only if information about them is known from physical considerations. This information must be stored as a property of the polynomial.

The Experimental Routines

The mathematical expression transformation routines will now be described in detail. Conclusions about the types of program organization which were needed for these routines have already been presented in the first part of the chapter. This detailed description will be useful to those who want to write similar programs. The transformation routines are written as a hierarchy of LISP functions. Some of the functions perform operations which are common to many of the transformations. These functions comprise a low level programming language. In addition, there are many functions which perform system and bookkeeping operations. To see the purpose of each of the function types, the processing of a sample command will now be traced through the system. A knowledge of the LISP programming language will be assumed for the remainder of the chapter.

Suppose the user were to type the command:

```
#'E←SIMPLIFY(DRV('X,1, 'X↑2 + '(X+1)↑2))#
```

the PDP-6 sends this command, as a list of characters, to the 7094. There, the LISP function APARSE, described in Chapter X, is used to parse the string of characters into a LISP expression. The above command becomes:

```
     (EASSIGN (EQUOTE E)(SIMPLIFY (DRV (EQUOTE X) 1
          (PLS (PWR (EQUOTE X) 2) (PWR (EQUOTE (PLS X 1) ) 2)))))
```

This expression is then evaluated in the LISP system. All mathematical expressions are kept on the disk. The function EASSIGN is one of several *system functions*. It will write the mathematical expression which is the value of its second argument on the disk as a file with name E FORM. The first name E has been pseudo-quoted with EQUOTE. EQUOTE transforms typed input expressions into internal form. It is one of several *functions for transforming expressions into equivalent forms for input or output*. Next, consider the evaluation of the arguments of DRV, the derivative function. DRV will take the fi st derivative with respect to X of its third argument. Evaluating the third argument, X and 2 are given to the function PWR, which is one of a group of *functions for performing the algebraic operations*. PWR[X,2]→(PWR X 2 NIL) Similarly EQUOTE [(PLS X 1)]→(PLS X 1 NIL), PWR[(PLS X 1 NIL), 2]→(PWR (PLS X 1 NIL) 2 NIL), and the final result is (PLS(PWR X 2 NIL) (PWR(PLS X 1 NIL) 2 NIL) NIL). The functions PLS and PWR will perform certain simplifications, such as PLS [X, (PLS A B NIL)]→(PLS X A B NIL) instead of (PLS X (PLS A B NIL) NIL). This reduces the growth of expressions because of simple redundancies. Next, the derivative function DRV is called. DRV issues a call to DIFF[X, (PLS X 2 NIL)(PWR (PLS X 1 NIL) 2 NIL)NIL)]. The function DIFF is one of several large routines for transforming the elementary functions. It would take up too much space if it stayed in core memory at all times. Therefore, DIFF is defined as a call to the system function GETFILE. GETFILE[DIFF, (X EXP)] first reads in the disk file DIFF LISP, which contains the DIFF routines. Then GETFILE evaluates DIFF[X, EXP]. Finally, GETFILE reads the disk file DIFF ERASE, which removes the DIFF routines from core memory. As stated above, DIFF is given the expression:

(PLS (PWR X 2 NIL) (PWR (PLS X 1 NIL) 2 NIL) NIL).

Before differentiating this expression, DIFF gives it to MKP, the function for putting the variables in all subexpressions on their property lists. MKP returns (PLS(PWR X 2 (X))(PWR (PLS X 1 (X)) 2 (X)) (X)). Now DIFF can look on the property list of an expression in order to see if it depends on the variable of differentiation. If it does not, DIFF will immediately return the answer, 0. For the example above, DIFF returns:

(PLS (PRD 2 X NIL) (PRD 2 (PLS X 1 NIL) (PLS 1 0 NIL) NIL) NIL).

This result is given to SIMPLIFY. This routine is used so often that it has been chosen to reside permanently in core memory. Therefore, no call to GETFILE is needed this time. SIMPLIFY finds that (PLS 1 0 NIL) $\rightarrow$ 1, then that (PRD 2 (PLS X 1 NIL) 1 NIL) $\rightarrow$ (PRD 2 (PLS X 1 NIL) NIL). Since this later expression is the argument of a PLS, it is expanded to (PLS (PRD 2 X NIL) 2 NIL). Thus, at the top level SIMPLIFY finds:

(PLS (PRD 2 X NIL) (PRD 2 X NIL) 2 NIL), which can be simplified by collecting terms. To do this, the routine must discover that two of the terms have the same literal factor. To compare these factors, SIMPLIFY uses the functions for computing the hash code number of a symbolic expression. If the hash code numbers are the same, SIMPLIFY assumes that the factors are equivalent. The final answer, (PLS 2 (PRD 4 X NIL) NIL) is now written on the disk by EASSIGN.

In addition to the five classes of functions above, there are also a few general purpose functions. The functions in each of these six classes will now be explained. Then the functions which comprise each of the transformation routines will be described. These descriptions use several special terms.

The expressions are represented in LISP in operator prefix form. The individual operators are:

(Note: most of the operators can take any number of arguments in the obvious manner.)

1. (PLS A B C NIL) $\equiv A+B+C$
2. (PRD A B C NIL) $\equiv A \cdot B \cdot C$
3. (FRT I J NIL) $\equiv \frac{I}{J}$
4. (PWR A B NIL) $\equiv A^B$
5. (DRV A B C D E NIL) $\equiv \frac{d^{B+D}}{dA^B dC^D} E$
6. (ITG D A B C NIL) $\equiv \int_A^B C \, dD$
7. (SUM A B C D NIL) $\equiv \sum_{A=B}^{C} D$

8. (EVL A B C D E NIL) $\equiv E \Big/_{A=B}^{C=D}$

9. (NAM A B C NIL) $\equiv C_{A,B}$

10. (F A B NIL) $\equiv$ F(A,B)

11. (NAM A B (F C D NIL) NIL) $\equiv F_{A,B}\ (C,D)$

12. (FTL A NIL) $\equiv A!$

13. (ABS A NIL) $\equiv |A|$

14. (DEL A NIL) $\equiv \frac{d}{dA}$

15. INF $\equiv \infty$

16. IDF $\equiv$ indefinite

17. PI $\equiv \pi$

18. *E $\equiv$ e

19. *I $\equiv$ i

20. NIL $\equiv$ undefined

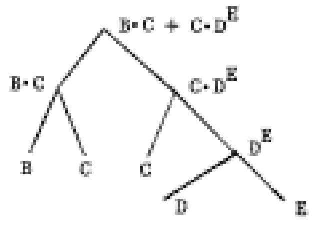
21. (XST I B C A NIL) $\equiv$

22. (IMS I B C A NIL) $\equiv$

23. (CND (A B NIL)(C D NIL) NIL) $\equiv \begin{matrix} A\to B \\ C\to D \end{matrix}$

The last element of each list is the <u>property list</u> of the expression. An empty property list is represented by NIL. When the operator is removed from an expression, the <u>argument list</u> remains. For example, the argument list of (PLS A B C NIL) is (A B C NIL). The <u>body</u> of an operator is the last argument in the list. For example, the body of (NAM A B C NIL) is C. The expression (F A B NIL) is called an <u>arbitrary function</u> of A and B. An expression such as

(PLS (PRD B C NIL) (PRD C (PWR D E NIL) NIL) NIL) forms a tree of subexpression as shown below:



When a function is said to go down the expression tree, it processes the nodes of this tree in the normal order for LISP.

Functions Which Put the Dependence Property on the Property Lists of All the Subexpressions

As was stated in the introduction, it is sometimes useful to put the variables in a subexpression on its property list. This list of variables will be called the dependence list of the subexpression. Subscripted function names are distinguished from variables as can be seen in the following examples:

```
                    (F X Y NIL) → (F X Y (X Y))
         (PLS A (F X Y NIL) NIL) → (PLS A (F X Y (X Y)((F A X Y))
(PLS A (NAM 1 (F X Y NIL) NIL)NIL) → (PLS A (NAM 1 (F X Y (X Y))
                                        ((NAM 1 F) X Y)) (A(NAM 1 F) X Y)).
```

The dependence lists are computed by MKP[X]. MKP goes down the expression tree to the atoms, which are left unchanged. Coming back up the expression tree, each argument list is given to MKP1.

MKP1 [ARGLIST, EXP, PLIST] goes down the argument list making a dependence list by taking the union of the variables on which each argument depends with the accumulated result, PLIST. There are five cases necessary to find the dependence of an argument. If the argument is a number, its dependence is NIL. For an atomic argument, the result is a list of the atom. If the argument is an arbitrary function, the result is the function name added to the argument property list. All other expressions except NAM bodies yield their property lists. As shown in the examples, NAM bodies yield the entire expression with the property list removed by RPLST and with arbitrary functions replaced by their names, combined with the dependencies of the NAM body function arguments.

Functions Which Compute the Hash Code Number of an Expression

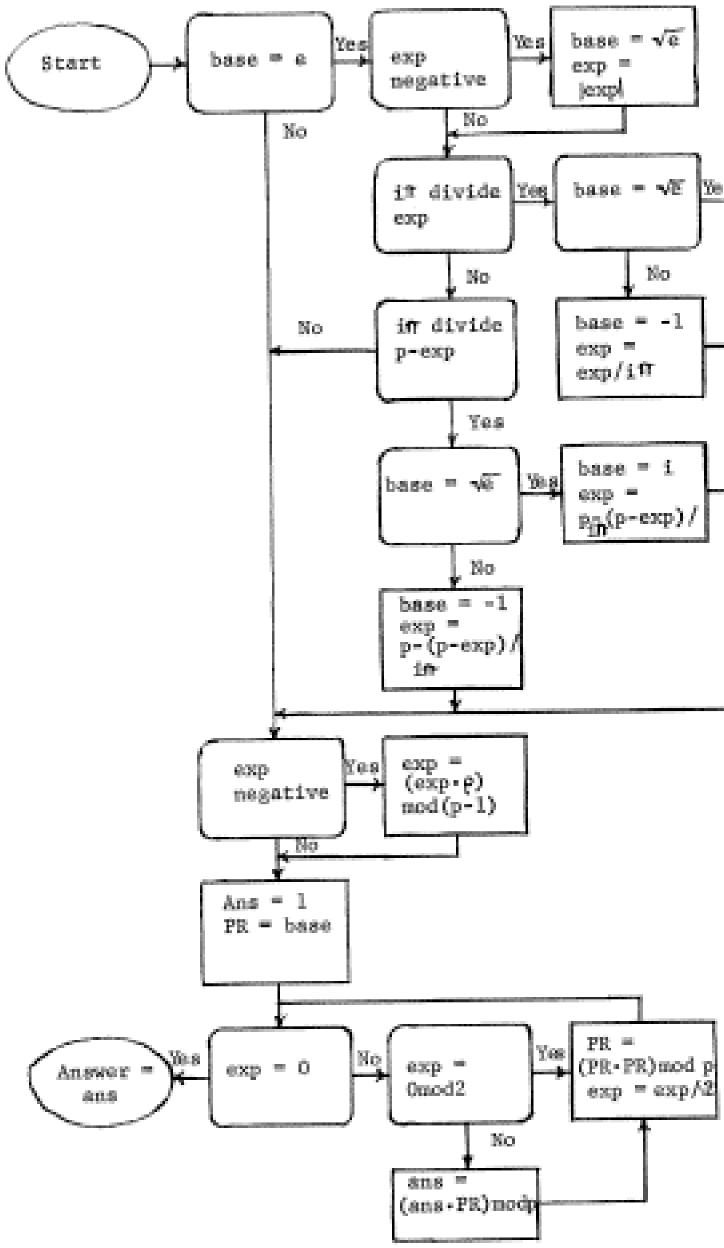
Chapter VII presents a scheme for assigning integers to expressions in such a way that equivalent expressions will be assigned the same integer. These integers are called hash code numbers because the infinite set of mathematical expressions is mapped into a finite set of integers by the assignment scheme. As explained in Chapter VII, if a^{x+2} is to have the same hash code number as $a^x \cdot a^2$, then a special hash code scheme must be used for exponents. For want of a better solution, exponents of exponents are treated with the base scheme. The current depth of exponentiation module 2 is assigned to the variable SIMPLEVEL. SIMPLEVEL is 0 for the base arithmetic, which was called F arithmetic in Chapter VII. SIMPLEVEL is 1 for the exponent or S arithmetic. The F and S arithmetic is performed by the functions FPLS, FPRD, FDVD, FPWR, SPLS, SPRD, SDVD, and SPWR. All of these functions except FDVD are coded in LAP for greater efficiency; the functions perform the following computations:

(Note: A and B are arguments and p is the finite field prime.)

FPLS: Answer = (A+B) mod p

FPRD: Answer = (A·B) mod p

FPWR: First a check is made to see if the base is equal to the integer assigned to e. If so, special code is entered so that $e^{ni\pi}$ will often hash into -1^n. In any case, a check must be made for a negative exponent which indicates that the finite field square root of the base is to be taken. Finally, the base is multiplied by itself in order to create the series base, base^2, base^4, base^{2^i} ... base^{2^n}. The answer is the product of all the terms in this sequence for which the corresponding bit of the exponent is a 1. The flow chart is:


Start
base = e
Yes
No
exp negative
Yes
No
base = √e
exp = |exp|
iπ divide exp
Yes
No
base = √e
Yes
No
base = i
exp = exp/iπ
base = -1
exp = exp/iπ
iπ divide p-exp
No
Yes
base = √e
Yes
No
base = i
exp = p-(p-exp)/iπ
base = -1
exp = p-(p-exp)/iπ
exp negative
Yes
No
exp = (exp·p) mod(p-1)
Ans = 1
PR = base
Answer = ans
Yes
exp = 0
No
exp = 0mod2
Yes
No
PR = (PR·PR)mod p
exp = exp/2
ans = (ans·PR)mod p

SPLS: Either argument can be positive or negative.

$A > 0$	$B > 0$	Answer = $(A+B) \bmod (p-1)$
$A > 0$	$B < 0$	Answer = $(-2A+B) \bmod (p-1)$
$A < 0$	$B > 0$	Answer = $(A-2B) \bmod (p-1)$
$A < 0$	$B < 0$	If $A+B$ is odd: Answer = $(A+B) \bmod (p-1)$
		Otherwise: Answer = $(\lvert A+B\rvert/2) \bmod (p-1)$

SPRD: Either argument can be positive or negative.

$A > 0$	$B > 0$	Answer = $(A \cdot B) \bmod (p-1)$
$A > 0$	$B < 0$	If $A \cdot B$ is even: Answer = $\lvert(A \cdot B) \bmod (p-1)\rvert/2$
		Otherwise: Answer = $(A \cdot B) \bmod (p-1)$
$A < 0$	$B > 0$	If $A \cdot B$ is even: Answer = $\lvert(A \cdot B) \bmod (p-1)\rvert/2$
		Otherwise: Answer = $(A \cdot B) \bmod (p-1)$
$A < 0$	$B < 0$	Answer = $-\ \rho\{[(A \cdot B) \bmod (p-1)]\} \bmod (p-1)$
		$\rho = 2n+2$ where $p = 16n+13$

SPWR: If the base is negative, then it is multiplied by $(2n+2) \bmod (p-1)$. It is then raised to the exponent in the same manner as for FPWR. First, the series base, base^2, ... base$^{2^k}$ ... is formed, where each term is the square of the preceding one, taken mod $(p-1)$. Then the answer is formed by multiplying together mod $(p-1)$ the terms k for which the kth digit of the exponent is a one.

SDVD: A is the numerator and B is the denominator.

```
A > 0     B > 0      A odd → B odd → Answer = $(A \cdot B^{-1}) \bmod (p-1)$
                                   T → Answer = $[-A \cdot (B/2)^{-1}] \bmod (p-1)$
                         T → B odd → Answer = $(A \cdot B^{-1}) \bmod (p-1)$
                                   T → Answer = SDVD[A/2,B/2]
A > 0     B < 0      Answer = $(2A \cdot B^{-1}) \bmod (p-1)$
A < 0     B > 0      A odd → Answer = $(A \cdot B^{-1}) \bmod (p-1)$
                         T → Answer = $(|A/2| \cdot B^{-1}) \bmod (p-1)$
A < 0     B < 0      Answer = $(|A| \cdot |B|^{-1}) \bmod (p-1)$
```

Where B^{-1} is computed as SPWR(B, 8n+3), since 8n+3 is $\phi(p-1)$, the number of integers less than p and relatively prime to p-1.

FDVD is computed as FPRD[A, FPWR[B, p-2]]. Frequently, one wants to perform an operation with either the S or the F functions depending on the SIMPLEVEL. For this reason, the A functions have been defined. These functions are APLS, APRD, ADVD, APWR, and ANEG. Their definitions are similar. For example:

```
APLS[A,B] ≡ (SIMPLEVEL = 0) → FPLS[A,B]
                   T         → SPLS[A,B]
```

ANEG[A] calls either FPRD[A, p-1] or SPRD[A, p-2].
This completes the explanation of the algebraic operations used in the hash code scheme. There are several functions which use these operations to compute the hash code number of an arbitrary expression.

The hash code number of any expression, X, is computed by the function ECODE[X, SIMPLEVEL]. ECODE goes down the expression tree until it reaches the individual variables and constants. It finds the hash code numbers for these and then computes the hash code number of each subexpression from the hash code numbers of its arguments. In going down the expression tree, the SIMPLEVEL is changed whenever the exponent of a PWR is entered or whenever the main connective of a subexpression is flagged with FCODE, indicating that it is a transcendental function. The code numbers of the individual atoms and numbers are computed by the function GETC[N]. GETC gives a floating point number to MKCD. In the case of a positive integer, it is only necessary to take APLS[integer, 0], or ANEG[-integer] for a negative integer. If N is a function, the answer is ADVD of GETC applied to the numerator and denominator. The remaining possible argument for GETC is a literal atom. GETC looks on the property list of the atom under the indicator CODE. If the atom has been coded before, or if it is *E, *I, or PI, its code number will be found. Otherwise, a random number is generated by RANNO and taken modulo p-1 by SPLS[random number, 0]. This code number is put on the property list of the atom and also returned as the answer. The floating point number hashing function, MKCD, hashes the number in three steps. First, the absolute value of the number is taken. Then the characteristic and fraction are separated, and the fraction is either multiplied or divided by 2*|characteristic|, depending on the sign of the characteristic. Finally, if the floating point number was negative, ANEG of the answer is taken.

Returning to ECODE, the method of computing the code number of an expression from the code numbers of its arguments, depends on the operator of the expression. As stated above, if the operator symbol has the property FCODE, this represents a function of one variable; the functions in this class which are presently coded are: FSIN, FCOS, FTAN, FSEC, FCSC, FCOT, FSINH, FCOSH, FTANH, FCOTH, FSECH, and FCSCH. These functions will be discussed individually later. They all compute a hash number from the hash number of their argument. If the operator symbol has the property NCODE, this property flags a function of one argument which is a list of the hash code numbers of the arguments of the operator. When a new operator is introduced, a function for computing its hash code can be invented and added to the system with NCODE. ECODE applies this function to a list of the argument hash numbers. ECODE calls APLS, APRD, and APWR for the operators PLS, PRD, and PWR respectively. In the case of PLS and PRD, ECODE uses ECODE1 to apply the binary operators APLS and APRD to the indefinite number of arguments of PLS and PRD. All other operators are handled by a call to FGFN. This function produces a result which depends on the hash numbers of the arguments and their order, but which has no invariant properties. Taking the hash number of the operator name as the initial accumulated answer, the accumulated answer is added in F to the hash number of the next argument, and this sum is raised in F to the arbitrary power 125786059 to form the new accumulated answer. This process is continued until all of the arguments have been used. This completes the discussion of ECODE. In conclusion, the formulas for the transcendental functions are:

FSIN $\frac{1}{2i}\left(e^{ix}-e^{-ix}\right)$

FCOS	$\frac{1}{2}\left(e^{ix}+e^{-ix}\right)$
FTAN	$u = e^{2ix}$ Answer = $\frac{1}{1}\cdot\frac{1-u}{1+u}$
FSEC	1/FCOS
FCSC	1/FSIN
FCOT	1/FTAN
FSINH	-1·FSIN(ix)
FCOSH	FCOS(ix)
FTANH	$\frac{\text{FSINH}}{\text{FCOSH}}$
FCOTH	1/FTANH
FSECH	1/FCOSH
FCSCH	1/FSINH

The operations in transcendental functions are performed in S arithmetic.

Functions for Algebraic Operations

The functions which perform algebraic operations can be divided into seven classes. Typically, the functions within a given class are defined for the same arguments, such as integers, rational numbers, or general expressions. Each class may also contain a few helping functions.

A. Integer Functions

Two functions accept integer arguments:

ABS[N] = |N|

FTLEXP[N] = N!

B. Rational Number Functions

The arguments for these functions can be either rational numbers, represented by (FRT M N NIL), integers, or floating point numbers. Since the functions for addition, multiplication, and division take either fractions or LISP numbers as input, they first send their arguments to DONUM for conversion to a standard form:

DONUM[A B] = (numerator of A, denominator of A, numerator of B, denominator of B)

Using this list, the three functions form the following quotients.

PLSNUM: $\dfrac{\text{NUM1} \cdot \text{DENOM2} + \text{DENOM1} \cdot \text{NUM2}}{\text{DENOM1} \cdot \text{DENOM2}}$

PRDNUM: $\dfrac{\text{NUM1} \cdot \text{NUM2}}{\text{DENOM1} \cdot \text{DENOM2}}$

DVDNUM: $\dfrac{\text{NUM1} \cdot \text{DENOM2}}{\text{DENOM1} \cdot \text{NUM2}}$

These results are then given to REDNUM to be reduced to lowest terms. If one of the original arguments was a floating point number, then of course, the answer will be a floating point number. If the denominator is zero, then REDNUM returns INF, representing ∞. Otherwise, REDNUM reduces the fraction to lowest terms by finding the g.c.d. of the numerator and denominator with GCD. The final member of this class is the exponentiation function PWRNUM. The exponent must be an integer. The special case 0^{-N} results in INF. Any other negative exponent results in the base being inverted with DVDNUM, and the exponent made positive. The remaining possibilities have a non-zero base with a positive exponent. If the base is negative, the sign of the answer is positive or negative depending on whether the exponent is even or odd.

C. Functions for Arbitrary Expressions

These functions combine expressions under the elementary operations. The expressions are not simplified, but simple redundancies are eliminated. EPLS[X, Y] and EPRD[X, Y] both call EFUN with appropriate functional arguments. EFUN performs the same operations for the additive and multiplicative group. EPRD must make the check for a zero argument, since $a \cdot o = o$ has no counterpart in the additive group. For INF·0 EPRD returns NIL and prints INDETERMINATE. EFUN[X, Y, W, NUFUN, Z, NUMP] has six arguments which have the following uses and values:

Name	Description	Value for EPLS	Value for EPRD
X	first expression		
Y	second expression		
W	operation name	PLS	PRD
NUMFUN	rational number function	PLSNUM	PRDNUM
Z	group identity	0	1
NUMP	group identity predicate	EZEROP	EONEP

If either argument is undefined, EFUN just returns (W X Y NIL). If either argument is the identity, EFUN returns the other argument. If either argument is infinity, the answer is infinity. If none of these conditions hold, then each of the two expressions is examined by EFUN1. EFUN1 constructs a number, U, and a term list, V. U is initially the identity and V is empty. If X is a number, it is combined with U. If X has the operator W, then its terms are placed on V, except that if the first term is a number, it is combined with U. Otherwise, X itself is placed on V. Y is treated in the same manner. EFUN then looks at this result. There are three possible cases. If the term list is empty, EFUN returns the number. If the number is the identity and there is only one term on the term list, this term is the result. Otherwise, the number is placed on the term list if it is not the identity, and the operator name W is added to form the answer.

The next function in this group is EPWR[X, Y]. There are quite a number of possible outcomes. If the exponent is one, the answer is the base. If either argument is undefined, the answer is (PWR X Y NIL). A zero exponent yields an answer of 1 unless the base is infinity. If the base is a fraction, then it is inverted if necessary, so that the numerator will be larger than the denominator. When the base is inverted, the sign of the exponent is changed. This canonical ordering of exponentiated fractions allows the current simplification program to reduce $(\frac{1}{4})^{-\frac{1}{2}} \cdot 2^{+\frac{1}{2}}$ to 2. If the base is a number and the exponent is an integer, then the result is computed by PWRNUM. If this is not so, but the base is a positive LISP number and the exponent is also a LISP number, then the result is computed by EXPT. Otherwise, the result is (PWR X Y NIL).

EDVD[X, Y] is defined as EPRD[X, EPWR[X, -1]] and ENEG[X] is defined as EPRD[X, -1]. EEQN[X, Y] is (EQN X Y NIL) and ESET[(X, $\cdots X_n$)] is (ESET X, $\cdots X_n$ NIL). EMIN[X] returns the smallest number on argument list X. It compares the arguments with the function LSSP described in the section on predicates.

ENLOG[X], the natural logarithm of X, can have several outcomes. If X is one, the answer is zero. If X is zero, the answer is $-\infty$. If X is ∞, the answer is ∞; and if X is e^A, the answer is A. Otherwise, the answer is (NLOG X NIL).

There is one more function in this group; EDRV[$X_1N_1 \cdots X_iN_i \cdots X_nN_n$), Y] expresses the N_i th derivative of Y with respect to X_i for each i. If Y does not have main operator DRV, then the answer is (DRV $\cdots X_i, N_i, \cdots$ Y NIL). Otherwise, EDRV gives the two arguments to EDRV1. EDRV1 gives Y, X_i, and N_i to EDRV2, for each i. EDRV2 goes through the list of differentiation variables on Y and adds X_i.

D. Predicates

There are eight predicates.

EENUMBERP[X]	Is X a LISP number, or (FRT N M NIL)?
ENUMBERP[X]	Is X a LISP number, INF, or (FRT N M NIL)?
EZEROP[X]	Is X a zero LISP number?
EONEP[X]	Is X a LISP number equal to 1.0 or (FRT N N NIL)?
EMINUSP[X]	Is X a negative LISP number or (FRT -N M NIL)?
EODDP[X]	Is X an odd positive LISP number?
GRTP[X, Y]	$X = Y \rightarrow$ NIL
	$X = \infty \rightarrow$ T
	$Y = \infty \rightarrow$ NIL

Otherwise, there is an error unless X and Y are LISP numbers, these are compared with GREATERP.

LSSP[X, Y] = GRTP[Y, X]

E. Functions Which Combine Expressions with Hash Code Numbers On the Front

There are four functions which combine lists of the form (hash number . expression) under addition, multiplication, and exponentiation. These functions are used by the simplification program. DPLS, DPRD, and DPWR all call DFUN with appropriate functional arguments. DPWR first checks to see if the hash of the base is zero. In this case the base and exponent are combined with EPWR and the result is prefaced with its hash code number. This special check is to avoid zero raised to a negative power.

DFUN[X Y AFUN EEFUN] is a function of the two arguments and the proper A and E functions for the given operation. DFUN combines the hash code numbers with the A function and the expressions with the E function. U is the new hash number and V the new expression. If the expression is NIL, then it is undefined and the hash of NIL is computed instead of U. The hash scheme breaks down for undefined operations. Next a check is made to see if U is 1, 0, or -1. In this case the expression is taken as equivalent to the hash number. (Note:) A hash number of -1 means that APLS[1, N] = 0). If none of these cases apply, the result is (U.V).

F. Functions Which Make It Possible to Evaluate the Expressions As LISP Functions

The operators have been given functional definitions so that they can be evaluated.

PLS[X, Y] = EPLS[X, Y]

PRD[X, Y] = EPRD[X, Y]

NEG[X] = ENEG[X]

DVD[X, Y] = EDVD[X]

EQN[X, Y] = EEQN[X]

NAM[X, Y] is an FEXPR. It evaluates its arguments and from the resulting list, $(X_1 \cdots X_n)$, forms (NAM $X_1 \cdots X_n$ NIL).

FUN[X, Y] is also an FEXPR. The first argument is the name of an arbitrary function. FUN evaluates its arguments and adds NIL to the end of the resulting list.

DRV[X, Y] is also an FEXPR. It evaluates its arguments and gives the resulting list to DRV1 which carries out the indicated differentiation. DRV1 gives each indicated variable of differentiation to DRV2, which calls DIFF in order to differentiate Y the required number of times.

G. Remaining Function

There is one more function which is needed to implement the convention that a set of one element and that element can be used interchangeably in commands. SETTER[X] returns X if its main connective is ESET, and (ESET X NIL) otherwise.

Functions for Input-Output

There are five operations performed to facilitate input-output.

1. Transformation of expressions from internal form before they are printed on the typewriter.
2. Transformation of typed input expressions into internal form.
3. Retrieval of the N^{th} subexpression of an expression which has been pointed at with the light pen.
4. Retrieval of an expression with the N^{th} subexpression removed.
5. Conversion of an expression retrieved by operation 3. or 4. into internal form.

Conversion of expressions for display is the subject of Chapter IX.

Expressions can be printed in a modified internal form with EPRINT. EPRINT[X] reads X FORM from the disk and removes the property lists from all subexpressions with RPLST. It gives the resulting expression to EPRINT1. EPRINT1 goes through the expression and gives all PLS, PRD, PWR, and EQN subexpressions which have more than one argument, to EPRINT2. EPRINT converts the expressions to internal operator form. For example, (PLS A B C) becomes (A PLS B PLS C).

If an input command contains a quoted expression such as '(A + B), this will be parsed into (EQUOTE (PLS A B)). EQUOTE is an FEXPR which returns its argument unevaluated, but transformed by EQUOTE1 and EQUOTE2. EQUOTE1 puts a null property list on every subexpression. It also transforms (NEG X) into (PRD -1 X NIL) and (DVD A B) into (PRD A (PWR B -1 NIL) NIL). EQUOTE2 removes redundant levels of PLS and

PRD expressions. For example,

(PLS A (PLS C D NIL) NIL) becomes (PLS A B C NIL).

GETSUB[X, N] returns the N th subexpression of X; the subexpressions are numbered in the order they would normally be evaluated by LISP. The subexpression is located by GETSUB1. GETSUB1 returns NIL if the count does not reach N, so it must list the answer to distinguish the answer NIL from failure. Therefore, GETSUB takes CAR of the result of GETSUB1. N is a free variable for GETSUB1; GETSUB1 decrements N at each new subexpression. When N is zero, GETSUB1 returns a list of that subexpression as the answer. If N is not yet zero and GETSUB1 reaches an atom or subexpression with operator ATOM, it returns NIL to indicate failure. If an expression has several arguments, GETSUB2 applies GETSUB1 to each argument in sequence, until N becomes zero or the arguments are exhausted. GETSUB is used to find a subexpression which is being pointed at by the light pen as described in Chapter IX. It may also be necessary to use the expression which contains this subexpression, perhaps replacing the subexpression with another one. Therefore, when GETSUB1 finds the N th subexpression, the list (SUBMARK M NIL) is put on the property list of the subexpression by altering the list structure with RPLACA. More than one subexpression may be marked by successive calls to GETSUB. The SUBMARK's will have increasing integers M until the constant INPNUM is reset to 1. This mechanism is used by the command REPLACE which is described in the discussion of commands.

The function REPLACESUB will replace each subexpression which has a SUBMARK structure on its property list, by that structure.

When a subexpression of expression E has been indicated with the light pen, then the structure E PFORM, which is produced by the picture compiler, is given to GETSUB. GETSUB finds the subexpression as described above. The PFORM subexpression is then transformed into internal form by APOFF. APOFF makes the following transforms on all applicable subexpressions:

1. (DVD A B NIL) $\rightarrow$ (PRD A (PWR B -1 NIL) NIL)
2. (NEG A NIL) $\rightarrow$ (PRD -1 A NIL)
3. (FUNCTION F B C NIL) $\rightarrow$ (F B C NIL)
4. (FUNCTION (NAM I F NIL) B C NIL) $\rightarrow$ (NAM I (F B C NIL) NIL)
5. (DRV 1 1 2 3 (F X Y NIL) NIL) $\rightarrow$ (DRV X 1 Y 3 (F X Y NIL) NIL)
6. (PAREN A NIL) $\rightarrow$ A
7. (ATOM A NIL) $\rightarrow$ A

APOFF1 and APOFF 2 perform transformation 5. APOFF3 performs transformation 4.

System Functions

There are fourteen specialized functions which communicate with the disk and the teletype. There are five types of disk files:

1. Files containing LISP functions which comprise a given mathematical transformation. These files have second name LISP.
2. Files containing LISP statements which delete from core the functions in a given transformation file. These have second name ERASE.

3. Files containing a single mathematical expression. These have second name FORM.
4. Files containing an intermediate form of a single displayed expression. These files are generated by the picture compiler to facilitate light pen reference to subexpressions. They have second name PFORM.
5. Files which contain scope instructions to display an expression. These files have second name DFORM.

There are functions which read, write and delete these files.

There is not enough space in core memory for all of the LISP functions which execute the commands. Therefore, the functions for a given operation, such as differentiation, are brought into core only while that step is being performed. In the case of differentiation, for example, the top level function DIFF[Y, X] is temporarily defined as a call to GETFILE[DIFF, (Y X)]. GETFILE reads file DIFF LISP, executes DIFF(Y, X), and then reads file DIFF ERASE which contains LISP commands to remove the differentiation routines from core and redefine DIFF as a call to GETFILE.

All mathematical expressions which have been defined in the system are written on the disk with second name FORM by a call to EASSIGN[Name, Expression]. The input command E←X becomes EASSIGN[E, X]. EASSIGN also deletes any old files with name E DFORM or E PFORM. An expression can be deleted by a call to EDELETE[E] which deletes E FORM, as well as E PFORM and E DFORM. The FEXPR FORM[E] has as value the expression in file E FORM. Unquoted names in input commands are given to FORM.

When a light pen reference is made, the function PFORMREAD[Name, N] is called. This function reads file Name PFORM and uses GETSUB to find the Nth subexpression. If an operation such as REPLACE is called, then the expression Name PFORM must be saved, so that a new subexpression can be substituted for the old one. To do this the call to PFORMREAD is replaced by a call to GETINSUB and the required PFORM is read into core and saved on the LISP constant INPFORM. GETINSUB then uses the expression on INPFORM instead of reading from the disk. When GETSUB locates the Nth subexpression, it alters the list structure of INPFORM to mark the place.

EDISPLAY[E] will type the display instructions for E. These instructions are preceded by a dollar sign, so that the PDP-6 will interpret them as a command, instead of typing them on the teletype. EDISPLAY first checks to see if there is a file E PFORM, indicating that the picture compiler has already been used to compute the displayed form of expression E. In this case EDISPLAY chains to the CTSS system command PRINT, printing file E DFORM. Otherwise, it is necessary to chain first to the file LISPP SAVED, which contains the picture compiler. This file reads E FORM and creates the corresponding PFORM and DFORM files. The DFORM file originally has name LISP LSPOUT, but this is changed to E DFORM with the CTSS RENAME command.

When a command is typed at the PDP-6, the string of symbols is given to the LISP function COMMAND. This function calls APARSE to parse the symbols into a LISP expression. It then evaluates this expression for its effect and returns an answer of NIL to indicate that the evaluation is complete.

A method of questioning the user has been provided with the function QUESTION[X, Y]. X is a list of expressions and Y is a question to be typed out, which may contain some of the expressions in list X. QUESTION first types the word QUESTION. Then, it generates a name for each expression in list X and writes the expression on the disk with this name. It also displays each expression. Next, it replaces the expressions in question Y with their names. The question is then typed. The user can make one of four responses. If he types GIVEUP, QUESTION calls ERROR. If he types ANSWERQ X, LISP expression X is returned as the answer. He may also type ANSWER X Y, in this case X Y is taken as a pair for EVALQUOTE and the result is returned as the answer. A special case check is made for ANSWER COMMAND (X). In this case string X is parsed with APARSE and evaluated to produce the answer. If the user does not type GIVEUP, ANSWER, or ANSWERQ, then his response is taken as a pair for EVALQUOTE and the result is typed out. QUESTION then types, "THIS QUESTION TO YOU IS PENDING", repeats the question and awaits a further user response.

There are three general purpose file reading functions. READF[X Y] opens file X Y, reads one list, then closes the file. FILESEND[X Y] reads one list from file X Y and prints the members of this list one at a time. WHOLEFILEREAD[X Y] returns a list of all the lists in file X Y.

General Purpose Functions

There are a number of functions which have proven useful in transforming the mathematical expression representations used in this thesis.

APND[X Y] forms an argument list from the argument lists X and Y. For example, APND[(A B NIL) (C D E NIL)] is (A B C D E NIL).

ARGLIST[X] If the next to last element of list X is an arbitrary function with arguments, or a subscripted arbitrary function with arguments, then the list of arguments is returned. Otherwise, NIL is returned.

DELETE[X Y] removes the first occurrence of Y from list X.

DEPEND[Y X] is true if expression Y depends on variable X. Either Y is an atom and X is equal to Y, or X is a member of the property list of Y. If X is a subscripted variable or function, then it will appear on the property list of Y in a special format, without arguments and with a null property list. In this case, X must be transformed into this format before it can be compared against the property list of Y.

FLOAT[X] X must be a LISP number or the form (FRT N M NIL). If X is a LISP number, it is floated by adding it to floating zero. Otherwise, the numerator and denominator are floated in this manner and then the quotient is taken.

LAST[X] returns the last member of list X.

MAPL[X, FN] is like MAPLIST except that it applies FN to CAR of the list X, and it does not apply FN to the last member of list X.

MATCH[EXP, PATT, CONST] is a very simple matching function. The expression EXP is matched against the pattern PATT in a two step process. If a match occurs, then the expression CONST is evaluated against a special ALIST, MALIST which is used during the matching step. In the matching step, MATCH2 compares EXP and PATT element by element, all elements must be the same until the ends of both expressions are reached or until the atom DOLLAR appears as an element on PATT. If this match succeeds, then the second part of the match is attempted. If PATT has a function on its property list, then the value of this function applied to EXP must be T. If both of these conditions are met, the match succeeds.

MKCLEANP[X] sets to NIL all subexpression property lists of expression X.

MKPLS[X] checks argument list X for three conditions:

$$(\text{NIL}) \rightarrow 0$$

$$(\text{A NIL}) \rightarrow \text{A}$$

$$(\text{A} \cdots \text{B NIL}) \rightarrow (\text{PLS A} \cdots \text{B NIL}).$$

MKPRD[X] checks argument list X for three conditions:

$$(\text{NIL}) \rightarrow 0$$

$$(\text{A NIL}) \rightarrow \text{A}$$

$$(\text{A} \cdots \text{B NIL}) \rightarrow (\text{PLS A} \cdots \text{B NIL}).$$

NILON[X] adds the element NIL to the right end of list X.

NTOL[X] returns the next to last element of list X.

NTOLSUBST[X, Y] substitutes Y for the next to last element of X.

NUMBER[X] is the same as LENGTH of X.

PUT[X Y Z] puts X on atom Y with indicator Z.

RANNO[ ] returns a random integer which is formed by multiplying the previous random integer by 3^{11}. The previous random integer is kept as the value of the constant RANDOMNUMBER. This random number generator is based on a note by Martin Greenberger.

REMPROPS[LIST PROP] removes property PROP from each element of list LIST.

RPLST[X] removes the property list from each subexpression of expression X.

SIMP3Z[FN Y] evaluates (FN Y).

STRIKE[X Y] removes from list X any element for which the predicate Y is true.

SUBSTA[CONST PATT EXP] applies MATCH to each subexpression of EXP. SUBSTA starts at the ends of the branches of the expression tree and works toward the top.

TWOOFF[X] removes the last two elements from list X.

UNIONLIST[X Y] adds to list Y any members of list X which are not members of Y.

Transformations

The implementation of each of the transformations in Chapter II will now be described. Conclusions about these transformations have already been presented in the chapter introduction. The most informative are SIMPLIFY and TRUNCATE. The use of light pen references is explained in the description of REPLACE.

ALLSUMEXPAND

ALLSUMEXPAND[X] uses SUBSTA to apply SUMEXPAND to each SUM subexpression in X. Since SUBSTA comes up from the bottom,

$$\sum_{i=0}^{1} \sum_{i=0}^{i} x^j$$ would be expanded to

$$\sum_{j=0}^{0} x^j + \sum_{j=0}^{1} x^j .$$

BRINGOVER

BRINGOVER[NAME, TERM] brings term TERM to the other side of the equation with name NAME. The term is indicated with the light pen. BRINGOVER is not very clever; the term must be one or more factors in a sum or product. In order to compute the answer, BRINGOVER needs both the term and the equation with a hole where the term was removed. Normally, the function PFORMREAD which evaluates a light pen reference returns the subexpression referenced; the rest of the expression is lost. To save the rest of the expression BRINGOVER has been defined as an FEXPR. It has two arguments, the first will evaluate to the name of the equation and the second to the term. First, BRINGOVER finds the equation name and reads the PFORM with that name from the disk. The constant INPFORM is assigned the PFORM as value. Now, before the second argument is evaluated the function name PFORMREAD is replaced with GETINSUB. Instead of reading the PFORM from the disk as PFORMREAD would do, GETINSUB uses INPFORM. The hole where the light pen reference was found is marked by altering the list structure of the subexpression property list. If the term is constructed from several light pen references, each will be marked. Having evaluated its second argument to obtain the term, BRINGOVER next calls REPLACESUB to replace the marked subexpressions in INPFORM with the lists (SUBMARK N NIL). N numbers the marked subexpressions. INPFORM is converted to standard internal form by APOFF and SUBMARK expressions with a number other than 1 are deleted from it by BRINGO1.

BRINGOVER now has the required term and equation with the hole marked by (SUBMARK 1 NIL). It first checks to make sure that it was really given an equation. If not, it just replaces the term. Otherwise, it checks for each of the following legal possibilities. Let X be the term to be moved.

$$X = Y \rightarrow 1 = YX^{-1}$$

$$Y = X \rightarrow X^{-1}Y = 1$$

$$AX = Y \rightarrow A = YX^{-1}$$

$$A + X = Y \rightarrow A = Y - X$$

$$Y = AX \rightarrow X^{-1}Y = A$$

$$Y = A + X \rightarrow Y - X = A$$

In each of these cases, it performs the required operation to produce the answer. Otherwise, the answer is produced by replacing the term in the hole.

COLLECT

COLLECT[EXP, SET] collects terms in the top level of expression EXP in powers of the variables in set SET. If the expression is an equation, it will collect terms on each side. When possible, COLLECT will combine terms involving summation. For example,

$ax^2 + \sum_{i=1}^{10} bx^i + d \cdot \sum_{j=5}^{15} cx^{j+2}$ will be collected on powers of x to

yield:

$$bx + (a+b)x^2 + \sum_{i=3}^{6} bx^i + \sum_{i=7}^{10} (b+d \cdot c)x^i + \sum_{i=11}^{17} d \cdot cx^i$$

This transformation is straightforward enough, but since there may be several variables on which to collect terms, and several factors within each term, as well as many intersecting ranges of summation, an inefficient algorithm could be very slow. Therefore, the data is rearranged into a special form which makes it easy to sequence through the collection variables, term factors, and summation ranges. Thus, the LISP functions are specialized to this routine. The complex data structures made the routine more difficult to debug.

The flow of control through the functions which comprise this routine is shown in Figure 1. The key data structure is shown in Figure 2.

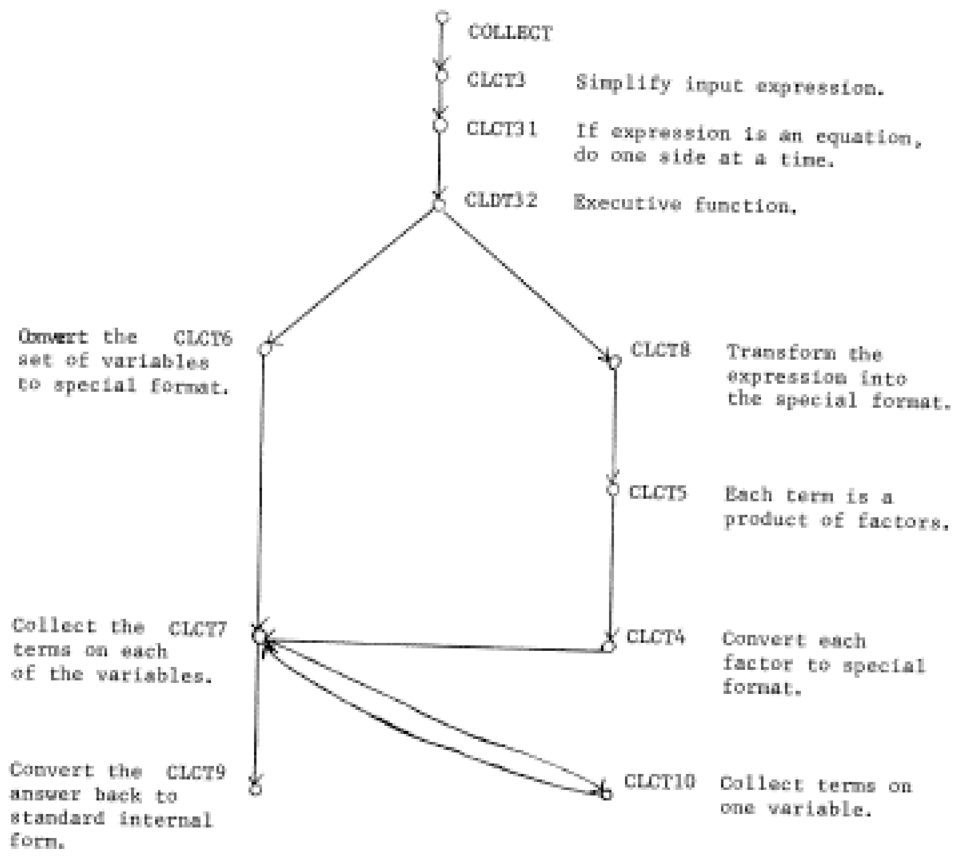

COLLECT
CLCT3
Simplify input expression.
CLCT31
If expression is an equation, do one side at a time.
CLDT32
Executive function.
Convert the set of variables to special format.
CLCT6
CLCT8
Transform the expression into the special format.
CLCT5
Each term is a product of factors.
Collect the terms on each of the variables.
CLCT7
CLCT4
Convert each factor to special format.
Convert the answer back to standard internal form.
CLCT9
CLCT10
Collect terms on one variable.

Figure 1

Flow of Program Control in COLLECT

The top level function COLLECT[EXP, SET] simply calls CLCT3[EXP, SET]. CLCT3 changes SET into a list of variables by removing the operator and property list; if a set of one element has been represented by the element itself, CLCT3 just lists the element. CLCT3 also simplifies EXP. CLCT3 gives EXP and the list of variables Y to CLCT31. CLCT31 checks EXP to see if its main connective is EQN. If it is, then the two sides are collected individually, otherwise, the entire expression is collected. The part to be collected is converted to a PLS term list and given to CLCT32. CLCT32 issues the function calls necessary to complete the collection of terms. First, the PLS term list X is converted by CLCT8. The conversion is shown by the example in Figure 2. CLCT8 takes the PLS terms one at a time; each of the PLS terms is converted to a PRD term list by CLCT5 and then given to CLCT4. The PLS term itself is added to the front of the answer produced by CLCT4 to form the list structure shown in Figure 1. To produce its result, CLCT4 checks each of the PRD terms to see if it has PWR or SUM for a main connective. If the main connective is PWR, a term of the form:

(hash of base, exponent, remaining PRD terms, preceding PRD terms) is constructed. If the main connective is SUM, several slightly more complex terms may be constructed. First, *I* is substituted for the running variable in the body of the SUM and it is converted to a PRD term list. The SUM body is then examined by CLCT41. If a term of the SUM body is a PWR and the base does not depend on *I* but the exponent does, then the exponent is evaluated by CLCT411. CLCT411 returns NIL unless the exponent is of the form *I*+N or N+*I*, N an integer, in which case it returns N. If CLCT411 returns NIL, then the factor cannot be collected and the machine asks the user if it should continue, ignoring this factor. If CLCT411 returns an integer, the terms in the SUM body PRD list before and after the term in question are concatenated together and made into a PRD. If the integer returned is not 0, that is we have (base^{*I*+N}), then a change of variables is made in the SUM so that we have (base^{*I*}) by substituting *I*-N for *I*. A term of the form:

(hash of base, (lower limit + N, upper limit + N), remaining PRD terms, preceding PRD terms including residue SUM body)

is then generated. If CLCT41 finds a term in the SUM body PRD list which does not depend on the running variable *I*, then if the term is a PWR a term of the form:

(hash of base, exponent, remaining PRD terms, preceding PRD terms including SUM with term in question factored out and running index changed to *J*)

is generated. If the term is not a PWR, a term of the form:

(hash of term, 1, remaining PRD terms, preceding PRD terms including SUM with term in question factored out and running index changed to *J*)

is generated. That is, the term is factored out of the SUM. If a term cannot be generated, then CLCT41 moves on to the next term in the SUM body PRD list. Returning to CLCT4, if it does not find the main connective to be PWR or SUM, then it forms a term:

(hash of term, 1, remaining terms in PRD, preceding terms in PRD)

When all terms have been examined, CLCT4 returns to CLCT8.

After the expression is transformed by CLCT8, the list of variables to be collected on is transformed by CLCT6. CLCT6 generates a structure of the form: ((hash of variable 1) (hash of variable 2) ... (hash of variable n)0), which will be called the <u>collection list</u>.

The transformed expression and list of variables are given to CLCT7. The purpose of CLCT7 is just to give the terms of the top level PLS one at a time to CLCT10, which puts them on the collection list in the appropriate place. The PRD list of each of the terms in the top level PLS has been converted by CLCT4 into a list of terms each of which begins with a hash number of a variable or expression and contains the information needed to rewrite the PRD with powers of that variable or expression factored out. CLCT10 goes completely through these PRD terms for each member of the collection list. If none of the PRD terms has a base which matches one of the collection variables, then the PLS term which these PRD terms make up is put at the end of the collection list. If more than one of the terms match, then the user is asked if it is OK to collect on the first variable or expression which matched. A mechanism is provided which lets him answer this question once for all. When the first match is found, the matching PRD list form and collection list entry are given to CLCT11. CLCT11 removes the hash number from the front of the collection list entry and then puts it back after the entry has had the new term added to it by CLCT111.

After the hash number has been removed from the front, the collection list is a list of the form

(X1 Y1 X2 Y2 ... Xn Yn) where

Xi is either an integer exponent or a list (N1, N2) giving a range of integer exponents, and Yi is the corresponding residue after factoring. Since part of a range Xi may match part of the range of the PRD list term, the remaining pieces must then be matched. CLCT111 tries all the current pieces against the current range Xi, adding the residue to Yi for each piece that matches and splitting Xi if part of it matches. CLCT11 then procedes to X(i+1) if any pieces of the range of the PRD list term remain. When Xn has been reached, all remaining pieces are put at the end by CLCT1111.

Once collection has been finished the collection list must be converted back to standard form. This is accomplished by CLCT9, which gives each term of the list to CLCT91 after removing the hash code number. CLCT91 generates a top level term for each pair Xi, Yi; a PWR if Xi is a single variable and a SUM if Xi is a range.

(A (PRD B C D NIL) (SUM J 0 10 (PRD E (PWR G J NIL) NIL) NIL) NIL)
is the term list to be converted to special format. There are three terms illustrating a single element, a product of factors, and a summation. The special format is:

```
                      Term             Decomposition of term into factors

First term   (    ((A NIL)          (#A      1   (NIL) NIL))          )
Second term       ((B C D NIL)      (#B      1   (C D NIL) (NIL))
                                    (#C      1   (D NIL) (D NIL))
                                    (#D      1   (NIL (C B NIL))      )
Third term        ((SUM *I* 0 10    (PRD     E   (PWR G *I* NIL) NIL) NIL) NIL)
                                    (#E      1       ((PWR G *I* NIL) NIL) NIL)
                                    (#G    (0 10)    (NIL) (E NIL))   )          )

           Hash code of factor base    Range of    Following and
                                       exponent    preceding cofactors
```

Figure 2

Conversion of terms for more efficient collection.

DEPENDENCE

DEPENDENCE[EXP] returns a set of the variables and arbitrary function names in EXP. It does this by making a set out of the property list of EXP, after MKP has been used to find the dependencies. If EXP is a number, DEPENDENCE returns NIL.

DELSUBST

DELSUBST[EXP, OLDDEL, NEWDELFREE] uses SUBSTA to search for subexpressions representing the first derivative with respect to the OLDDEL variable. For example, OLDDEL might represent $\frac{d}{dx}$ and SUBSTA might find $\frac{d}{dx}(a+b)$. Then if NEWDELFREE represented, for example, $c\frac{d}{dy} + e\frac{d}{dz}$, SUBSTA would use DELSUBST1 to make the substitution $\frac{d}{dx}(a+b) \to c\frac{d}{dy}(a+b) + e\frac{d}{dz}(a+b)$. This is one of the few uses of the DEL operator, (DEL X NIL) $\equiv \frac{d}{dx}$.

DRV

The operator DRV was described in the section on algebraic functions. There it was explained that DRV[X, 2, Y, 1, Z] would use DIFF to differentiate Z, first with respect to Y, and then twice with respect to X. The function DIFF[EXP, VAR] will now be described; DIFF differentiates EXP once with respect to VAR. The routine uses MKP to compute the dependence of subexpressions. Then, as DIFF comes down from the top, it can immediately replace a constant subexpression by zero. This method is certainly faster than simplifying a completely differentiated constant. However, it would be faster to work up from the bottom without the call to MKP. Since, MKP can be used in several routines, the method used here requires less program. It is very easy to specify derivatives with patterns and so they have been used here for the transcendental functions. The length of this routine is primarily the result of checks to reduce generation of redundant expressions.

DIFF differentiates an expression with respect to a given variable. The top level function, DIFF(Y, X), gives the FORM Y and variable X to DIFF2. DIFF2 makes one pass down to the ends of the branches of the expression tree, rewriting the expression as it goes. The transformation performed on each subexpression is a function of its main connective, the dependence of the subexpression and each of its arguments on the variable of expansion, and in some cases, the main connective of one of the arguments. However, the derivative of the expression depends on the derivatives of its arguments only in the sense that if some of the derivatives of the arguments are zero, the result is collapsed to avoid an explosive growth of redundancies. The various cases will now be discussed one at a time, in the order in which the expression is tested.

The derivative of a constant is zero. At the end of a branch, the expression must be the variable of expansion with derivative 1. If the operator has property DIFF, this is a pattern into which single argument of the function is substituted for *Y*, and which is then multiplied by the derivative of the argument. For example, SIN has the pattern (COS *Y* NIL), so that (PRD (COS X NIL) 1 NIL) would be formed as the derivative of (SIN X NIL). An arbitrary function initiates a call to DIFF8. DIFF8 forms a PLS term list. A term of the PLS is formed for each dependent argument of the function. DIFF8 forms the derivative of the function with respect to this dependent argument and multiplies this result by the derivative of the argument. The derivative of the body of a SUM or XST is taken, and all other arguments of these operators are left unchanged. PLS generates a call to DIFF3. DIFF3 forms a PLS term list of the derivatives of all the dependent arguments of the PLS. PRD initiates a call to DIFF4; DIFF4 also forms the term list of PLS. Each dependent term of the PRD is differentiated and multiplied by its cofactor to form a term of the PLS. PWR changes a^b into $\dot{a}\cdot(b\cdot a^{b-1}) + a^b\cdot(\log a)\cdot\dot{b}$. The integral sign is removed from an indefinite integral with respect to the variable of differentiation.

(A convention here). Otherwise, $\int_a^b f dy$ becomes

$$\dot{a}\cdot(-f(a)) + \dot{b}f(b) + \int_a^b \dot{f}(x)dx$$

where the third term is replaced by zero if f is independent of x. In the case of DRV, if the DRV body is an arbitrary function or a subscripted arbitrary function, then DIFF8 is used to apply the chain rule as above. Otherwise, EDRV is used to add the new variable to the DRV list. Both arguments of EQN are differentiated, as are all the arguments of ESET. A DRV structure is set up around TMS. The second argument of all pairs in a CND is differentiated by DIFF6. For NAM, a check is first made to see if the expression is equal to the variable of differentiation. If this is not the case, the expression can be dependent only if it is a subscripted function. The expression is thus given to DIFF8 for expansion by the chain rule. Any other permanent main connective does not have a derivative defined and will cause a call to ERROR.

DRVDO

DRVDO[EXP, VAR] carries out all indicated derivatives with respect to VAR in EXP. It uses DIFFDO1 to search out all DRV subexpressions. Each of these is given to DIFFDO2, the innermost ones first. DIFFDO2 examines each variable of differentiation on the DRV subexpression to see if any of them are VAR. If so, DIFFDO2 gives the DRV body, VAR, and the number of times, N, the body is to be differentiated with respect to VAR, to DIFFDO3. DIFFDO3 then issues N calls to DIFF. If there are variables of differentiation other than VAR, then these remain in a DRV subexpression.

DRVFACTOR

DRVFACTOR[EXP DRVFREE NFREE] uses SUBSTA to locate DRV subexpressions which indicate differentiation with respect to DRVFREE. It applies DRVFACTOR1 to each of these. DRVFACTOR1 searches down the list of variables of differentiation for DRVFREE. Then if the number of indicated derivatives with respect to DRVFREE is greater than NFREE, DRVFACTOR2 goes to the end of the argument list to form the new DRV subexpression. The resulting transformation is:

```
     (DRV ··· DRVFREE, (NFREE + M) ··· Y NIL) →
               (DRV ··· DRVFREE, M ··· (DRV DRVFREE NFREE Y NIL) NIL)
```

DRVZERO

DRVZERO[EXP, VAR] uses SUBSTA to substitute zero for each DRV subexpression in EXP which indicates differentiation with respect to VAR.

EVALUATE

EVALUATE[EXP, SET] searches expression EXP for variables and functions which can be evaluated using the equations in set SET. Only equations whose left hand side has one of the following forms are acceptable.

1. F
2. Fi, ··· ,j
3. F(X, ··· ,Y), matches F as well
4. Fi, ··· ,j (X, ··· ,Y), matches Fi, ··· ,j as well

The subscripts and function arguments can be either numbers or literals; if literals, they are taken as free variables to be bound. EVALUATE4 checks the equations and throws out unacceptable ones. Next, EVALUATE1 searches EXP for instances of variables or functions which can be evaluated with the equations. Suppose, for example, that EXP was a+x and SET contained the equations x=y and y=b, then the search by EVALUATE1 would locate x and substitute y, yielding a+y. But further evaluation can now be done with the second equation. Therefore, whenever an evaluation is made, the variable MARKFREE is set so that EVALUATE1 will make another pass through EXP. EVALUATE1 gives EVALUATE2 EXP and one equation at a time. EVALUATE2 goes through EXP, calling EVALUATE5 to match each subexpression with the left side of the equation. EVALUATE5 checks for each of the four cases given above. It uses EVALUATE6 to bind subscripts or arguments and substitute the values in the right side of the equation. Of course, for a match to occur the number of variables or subscripts must be equal.

EXCHANGE

EXCHANGE[EXP] will interchange the arguments of the main connective of EXP, if the main connective is binary.

EXPAND

EXPAND[EXP] goes through the expression EXP applying the distributive law. It expands sums raised to a positive integer power and distributes the derivative or integral of a sum. For historical reasons, the functions are named MULT. EXPAND calls MULT2[EXP]. A special data form has been used, the resulting gain in efficiency is probably not worth the special programming required. If EXP is an equation, MULT2 handles each side separately; otherwise, the entire expression is handled. MULT2 gives the parts to be handled to MULT3, which returns a PLS term list at the top level. MULT2 makes this list of terms into a PLS and simplifies it. MULT3 recurs down the expression tree if the expression has PLS, PRD, DRV, ITG, or PWR for its main connective. All of the arguments of PLS, PRD, and PWR, and the main expressions of DRV and ITG are treated in turn by MULT3. Expressions with any other main connective are listed and returned as the value of MULT3. Coming back up the expression tree each function receives as arguments and returns for a value, a list of terms which are arguments of a PLS. Thus, at the end of each branch of the expression tree is a list with a single term. These lists are combined in a manner appropriate for the main connective which the function represents; MULT3 gives PLS to MULT4, PRD to MULT5, DRV and ITG to MULT6, and PWR to MULT7. In the case of DRV and ITG, MULT3 uses MULT31 to get a list of the arguments other than the main expression and passes this list along to MULT6 as well.

MULT4 receives a list whose members are lists of PLS terms, with a NIL at the right end. Since MULT4 represents PLS, it just appends these lists together to form its answer.

MULT5 also receives a list whose members are lists of PLS terms with a NIL at the right end. It has a second argument which is initially NIL and is used to build up the answer. This second argument is a list of PLS terms. MULT5 uses MULT51 to multiply each member of its first argument by the partial answer represented by the second argument. MULT51 multiplies each term of one of its arguments by all the terms of the other. The resulting list of terms forms the answer for MULT51. To do this, MULT51 uses MULT511 to multiply each member of a list of terms by a single term. Each of these products is simplified to prevent expression growth.

MULT6 makes an integral or derivative subexpression out of each member of the list given it by replacing the member with the structure:

("first part of old list" "member" NIL).

MULT7 handles exponents; it distinguishes three cases.

If the list of terms which comprise the base argument has a single member, this member is raised to the exponent by EPWR. If the exponent is a positive integer, then the base is multiplied out, using MULT71. Otherwise, a single term is formed which represents the terms in the base as a PLS subexpression raised to the exponent. MULT71 multiplies the base by itself by using MULT51. It builds up the answer as its first argument. It decrements the exponent, which is the third argument, to count the required number of multiplications.

FACTOROUT

FACTOROUT[EXP, X, Y] is defined as an FEXPR so that it can have an optional third argument. The function will factor X from each term of expression EXP. If the third argument Y is present, Y will be substituted for the term factored out. For example, FACTOROUT[A + X, X, Y] yields $Y \cdot \left[\frac{A}{X} + 1\right]$, while FACTOROUT[A + X, X] yields $X \cdot \left[\frac{A}{X} + 1\right]$. FACTOROUT supplies FACTOROUT1 with X for a third argument if Y is not given. FACTOROUT1 will do each side of an equation separately. To factor X out, FACTOROUT1 just uses MAPL to divide each term by X and simplify it.

GROUP

GROUP is used for the light pen selection of certain arguments of a commutative operator which are to be handled as a group. For example, one might select the a and b from the expression a+c+b+e. These expressions might be replaced by d to yield d+c+e. The first argument of GROUP will evaluate to the name, X, of an expression. The second argument will evaluate to a set of terms or factors in a subexpression of X with a commutative operator. To determine what operator has been grouped, GROUP uses GROUP1 to search through the entire expression for an expression which has been marked by GETSUB. This marked expression will be an argument of the operator in question. In order to do this, GROUP must save X PFORM for GROUP1. It does this in the standard way. The first argument is evaluated to find the name of the expression. Then, the constant INPFORM is set equal to the corresponding PFORM. Next, the second argument is evaluated to find the set of grouped terms; during the evaluation GETSUB will mark INPFORM. INPFORM can then be given to GROUP1.

If INPFORM is not NIL when GROUP is entered, then some operation, such as REPLACE, contains this call to GROUP in one of its arguments. This function will need the marked INPFORM generated by GROUP. Otherwise, INPFORM is set to NIL in order to regain free storage.

LEFT

LEFT[EXP] returns the first argument of EXP.

LIMIT

LIMIT[EXP, X, N] substitutes N for X in EXP. However, if this substitution in a subexpression such as $a \cdot b \cdot c$ should result in $a \cdot 0 \cdot \infty$, then LIMIT will apply L'Hopital's Rule and evaluate $a \cdot \dfrac{\text{LIMIT[b', X, N]}}{\text{LIMIT[(1/C)', X, N]}}$. LIMIT simplifies EXP and calls LIMIT1[EXP, X, N]. LIMIT1 goes down the expression tree until it reaches the end of a branch, where it substitutes N if the branch end is X, or until it reaches a PRD subexpression. The factors of the PRD are then sorted on the program variables W, N, D, depending on whether their limit is non-zero finite, zero, or infinity. When all the factors have been sorted, LIMIT1 checks for the cases:

$$W \cdot 1 \cdot 1 \rightarrow W$$

$$W \cdot 0 \cdot 1 \rightarrow 0$$

$$W \cdot 1 \cdot \infty \rightarrow \infty$$

$$W \cdot 0 \cdot \infty \rightarrow W \cdot \frac{\text{LIMIT[N']}}{\text{LIMIT[D']}}$$

MULTIPLYTHROUGH

MULTIPLYTHROUGH[EXP, X] will multiply an expression or each side of an equation by X. If the expression or equation side to be multiplied by X is a PLS, MULTIPLYTHROUGH will multiply each term by X. In all other cases, it will just multiply EXP by X. The function uses EPRD for multiplication.

NEWNAME

NEWNAME uses the LISP character handling functions to generate a new atom of the form Fn, where n is an integer. The integer n is kept as the value of the constant NAMENUMBER. NAMENUMBER is incremented each time NEWNAME is called.

NORMPOLY

NORMPOLY[EXP, X] uses SUBSTA to search for all PLS subexpressions of EXP. Then NORMPOLY2 factors a power of X out of each term in the subexpression so that it will be a polynomial whose lowest power in X is zero. For example,

$$ax + bx^2 \rightarrow x(a + bx)$$

$$\frac{a}{x} + bx^2 \rightarrow \frac{1}{x}(a + bx^3)$$

Since SUBSTA comes up from the bottom,

$$ax + c(dx + ex^2) \rightarrow x(a + c(d + ex)).$$

NORMPOLY1 finds the power of X to be factored out by taking the minimum of the lowest power of X in each term. The lowest power of X in each term is computed by NORMPOLY4. If the term is a PRD, NORMPOLY4 uses NORMPOLY3 to add the lowest powers of each factor. The lowest power of X is of course 1, and the lowest power of X to a number is the number. All other expressions are assigned a lowest power of zero.

REPLACE

REPLACE[X, Y, Z] replaces the subexpression Z, which has been indicated with the light pen in the expression named X, by the expression Y. If the evaluation of Y contains a light pen reference to X, and Y is to be replaced at this same spot, then Z can be 'HOLE. Furthermore, if Y is 'NIL, then Z will just be deleted from X. Obviously, REPLACE must obtain the expression named X, with the subexpression Z marked by a light pen reference. To do this REPLACE uses a method common to all functions which need as an argument the expression in which a light pen reference has been made. REPLACE is defined as an FEXPR. It evaluates its first argument to get the name X of the expression. Then it reads from the disk the expression X PFORM. X PFORM is generated from X by the picture compiler, and it has the same subexpression structure as the displayed form of X. The constant INPFORM is assigned X PFORM as value. The light pen reference count INPNUM is set to zero. Next, REPLACE evaluates its second argument with the function PFORMREAD, which would read X PFORM from the disk in order to return a subexpression indicated by the light pen, replaced by GETINSUB, which will use INPFORM instead. Each light pen reference is marked in INPFORM by GETSUB, which is called by GETINSUB. GETSUB alters the list structure to mark it and uses INPNUM to count the subexpressions. When the second argument is evaluated, REPLACE has the expression Y and on INPFORM the expression named X with the references marked in it. If Z is HOLE, then REPLACE justs substitutes Y into X at a subexpression reference. If Z is not HOLE, then Z contains the light pen reference indicating where Y must be substituted. A new X PFORM is read in and Z is evaluated.

Now X has the proper subexpressions marked. X is converted from a PFORM to internal form by APOFF; the Nth marked subexpression is replaced by (SUBMARK N NIL). It remains to substitute Y. It is assumed that if there is more than one SUBMARK expression, they are all arguments of the same operator. REPLACE uses SUBSTA to locate this operator. REPLACE2 checks each argument of an operator to see if it is a SUBMARK. When an operator containing a SUBMARK argument is found, REPLACE1 replaces the first SUBMARK with Y and then REPLACE3 deletes the remaining SUBMARKS. If Y is NIL, then all SUBMARKS are deleted.

RIGHT

RIGHT[EXP] returns the second argument of EXP.

SIMPLIFY

SIMPLIFY starts at the ends of the branches of the expression tree and works toward the top, performing the following simplifications:

$$0 \cdot a \rightarrow 0$$

$$1 \cdot a \rightarrow a$$

$$0 + a \rightarrow a$$

$$a + (b + c) \rightarrow a + b + c$$

$$a \cdot (b \cdot c) \rightarrow a \cdot b \cdot c$$

$$a^n \cdot a^m \rightarrow a^{m+n}$$

$$i \cdot a + j \cdot a \rightarrow (i+j) \cdot a$$

$$i \cdot a + j \cdot (a+b) \rightarrow (i+j) \cdot a + j \cdot b$$

$$(a \cdot b)^m \rightarrow a^m \cdot b^m$$

$$\int -i \cdot a \rightarrow -1 \cdot \int i \cdot a$$

$$\frac{d}{dx} -i \cdot a \rightarrow -1 \cdot \frac{d}{dx} i \cdot a$$

where i and j are rational numbers. Furthermore, all indicated arithmetic operations involving integers, floating point numbers, and fractions are carried out. The above operations require the comparison of expressions for equality. This is accomplished by hash coding the expressions and comparing the resulting hash code numbers. Additional simplification is also achieved when an expression which is identically equal to zero or one is replaced by its hash code number.

Although the simplification program is not long, the use of the hash code scheme and several other schemes to improve efficiency make it tedious to read. Therefore, the sources of difficulty in each scheme will now be discussed.

Generation of a hash code number for a complex argument is moderately expensive. Therefore, it would be desirable to generate each hash code only once. However, the storage of a hash code for each subexpression would take up a significant fraction of the available storage. A decision was made to save the hash code numbers only at the current level of simplification. The hash code number of a subexpression is computed from the hash code numbers of its arguments. The hash code numbers of the arguments are then discarded. In the expression $d + (b+c)$, it would be incorrect to compute the hash code of $(b+c)$ and discard those of b and c, since the next operation is also addition and requires the comparison of d against b and c in an effort to collect terms. To solve this problem, the next higher operator is made available to the program. This is normally adequate, but in the expression $e \cdot (b-b+d^2)$, the next higher operator from d^2 is +; therefore, the hash code number of d will be discarded and the hash code of d^2 computed. Then, when $b-b+d^2$ simplifies to d^2, the hash code number of d must be recomputed for comparison with e. This is hopefully not a frequent situation.

If an expression is identical to zero or one, then it is probable that its hash code number will be zero or one. Therefore, whenever a new subexpression is formed, its hash code should be checked. If it is zero or one, the subexpression is taken as the same. The D functions described earlier perform this operation. The hash code number for each subexpression is kept on the front of it. The D functions combine the hash code numbers and the expressions, perform the identity checks, and then put the resulting hash code number on the front of the resulting expression.

A further complication in the hash code comparison is that the numbers must also be compared with the appropriate group inverse. For example, the code of $c\cdot(a-b)$ is the negative of the code for $c\cdot(b-a)$. But the expression $c\cdot(b-a) + c\cdot(a-b) + d$ should be simplified to d. Similarly, since $\cosh^2 x - \sinh^2 x = 1$, $(\cosh x - \sinh x)\cdot(\cosh x + \sinh x)\cdot d$ should be simplified to d. In fact, the only frequent comparison of multiplicative inverses would be in expressions such as $(\frac{1}{2})^{+\frac{1}{2}}\cdot 2^{-\frac{1}{2}}$. Since it is expensive to compute the multiplicative inverse, this has been solved by representing exponentiated fractions with the larger integer as numerator. Thus, $(\frac{1}{2})^{+\frac{1}{2}}$ becomes $2^{-\frac{1}{2}}$.

Finally, two types of arithmetic are used in the hash code scheme; the base arithmetic and the exponent arithmetic. This was explained in the discussion of the functions which compute the hash code numbers. For want of a better solution, exponents of exponents are coded in the base arithmetic. Therefore, as SIMPLIFY goes through the expression it must keep track of the current level of exponentiation modulo 2. In this sense, the addition of exponents of like factors in a product is not the dual of the addition of coefficients of like terms in a sum.

A mechanism has been added to insure that subexpressions represented by the same list structure will be simplified only once. After a subexpression is simplified, the list structure is altered so that other references to the subexpression will encounter the simplified form. The expression is marked "already simplified". The mark also gives the level of exponentiation for which the hash code number of the simplified expression has been computed. If the current reference has the other level, a new hash code number must be computed.

Combining terms or factors in a sum or product is a form of sorting operation. However, only an exhaustive search is used here. In order to speed up the search, the terms are combined in a special format. Whenever the operator at the level above is PLS and the current level is PLS, or the operator above is PRD or PWR and the current operator is PRD or PWR, then the sort format is passed up without conversion to standard internal form. During the combination of terms a numerical term is kept separate, for any number can be combined with another number, but only non-numerical terms with the same base can be combined. Non-numerical terms are compared by the hash code numbers of their bases. Unfortunately, it may result that the combination of two non-numerical terms results in a numerical term. For example, $2^{\frac{1}{2}}$ is considered non-numerical, but $2^{\frac{1}{2}}\cdot 2^{\frac{1}{2}}$ results in 2. Numerical terms generated in this manner are combined with the number which is kept apart when the sort format is converted to standard internal form. Conversely, in the expression $(2a)^{\frac{1}{2}}\cdot b$, the number 2 must be considered a non-numerical term after it is exponentiated.

The functions which carry out these operations can now be described. The top level function, SIMPLIFY, calls SIMP2. SIMP2[SX, Y, SIMPLEVEL] is a function of the expression to be simplified, the operator of which this expression is an argument, and the simplification level, that is, whether F or E arithmetic should be used in computing the hash code number of the expression. Going down the expression tree, SIMP2 must look for four conditions. The end of a branch of the expression tree is reached whenever an FRT operator or an atom is encountered. This expression is given to SIMP3, the function which handles the upward pass. If a PWR operator is encountered, then the SIMPLEVEL is logically OR'ed with 1 when SIMP2 is applied to the exponent. There is a similar step for the arguments of transcendental functions. It is also possible that the expression will have the form: ((hash·expression)·N), where N is the simplevel. SIMP2 checks for this structure by testing if CDR of the structure is atomic. This structure is set up whenever an expression is simplified, thus, if two subexpressions are in fact the same list structure, then when the first one has been simplified, SIMP2 will find the above structure when it reaches the second. It is possible that the two expressions do not occur at the same simplevel, therefore, the simplevel N at which the hash number was computed is part of the structure. If N is equal to the current simplevel, CAR of the structure is returned as the answer by SIMP2. Otherwise, the hash number is first replaced by one calculated for the current simplevel. In all other cases, SIMP2 goes on down the expression tree without a change in simplevel.

Coming back up the expression tree, SIMP3 checks for nine possibilities. SIMP3 has three arguments X, Y, and Z, the current operator, the expression term list, and the operator of the expression of which the current expression is an argument. If the current operator is flagged with FCODE, then it is a transacendental function. No simplification is done on the expression. The hash number is computed from the hash number of the argument by the FCODE function. If the resulting hash number is zero or one, then the expression is replaced with the hash number in the result by SIMP33. The result consists of the expression with the hash number on the front of it. If the current operator is flagged with NSIMP, then the list of arguments is given to the NSIMP function and the list of corresponding argument hash code numbers is given to the NCODE function. Since the input argument list contains the arguments with the hash code number of each on the front of it, the hash numbers and arguments are split into separate lists by the functions SIMP34 and SIMP31. The results of the NCODE and NSIMP functions are given to SIMP33 as above. The operators PLS and PRD cause SIMP3 to call SIMP5 with arguments appropriate to addition or multiplication. The arguments of SIMP5 and their possible initial values and purposes are:

X	term list of expression to be simplified
Y	operator above
SU	0·0 or 1·1, appropriate identity and hash code
V	NIL, list of collected terms
FNW	DPLS or DPRD, appropriate group operator
FNZ	DPRD or DPWR, appropriate group operator
R	PLS or PRD, appropriate group operator

SIMP5 removes each term from list X and takes one of six courses of action. The term may be:

1. A number. (M·N) The number is added to SU with FNW.
2. An expression having the same main connective as the current level and left in special sorting format: ((sort format).(M·N)) (M·N) is combined with SU by FNW. (sort format) is combined with V by SIMP51, using FNW, FNZ, and R.
3. An expression having the same main connective as the current level, but in standard form. The term is converted to sort format by SIMP54 and then given back to SIMP5, where it satisfies case 2.
4. A PWR and R is PRD. The term is converted to single term sort format by SIMP55 and given back to SIMP5 where it satisfies case 6.
5. A PRD with a numerical coefficient when R is PLS. The term is converted to single term sort format by SIMP55 and given back to SIMP5 where it satisfies case 6.
6. Anything else. The term may be in single term sort format already if the level below is PWR or if the term has been converted by SIMP55. Otherwise, it is listed with (1·1) to make a single sort term with a coefficient of one.

A more detailed explanation of the data formats will now be given along with a description of the functions auxiliary to SIMP5. Collecting terms in a product or sum is carried out using a special format. A list of the terms is formed; each member of the list has the form:

((code·base) (code·exponent))

Terms to be added to the list are put into this form; in the case of addition, the exponent is the numerical coefficient of the term. The code number of the base of a new term is then compared with the code number of each of the old terms on the list. If a match is found, the exponents are added. Otherwise, the new term is put at the end of the list. Thus, functions are needed to convert expressions into sorting format and from sorting format into normal form, and to add new terms to the sorting list. The function SIMP51 adds a list of new terms in sorting format to the sorting list. It gives each term to SIMP52. SIMP52 makes the comparison of bases described above. SIMP53 transforms terms from sort format into normal form. SIMP53 has as arguments the sort expression, the group operator, and the exponent operator. The base and exponent from the sort list are given to the exponent operator which constructs a term in normal form with its hash number on the front. If this term is a number, it is combined with SU in SORT5. If this is the last term on the sort list, then SIMP53 returns a list of the hash number, the converted term, and NIL. Otherwise, SIMP53 combines the hash number with that on the front of the list for the rest of the converted terms and adds the new term to this list, putting the new hash number on after it. SIMP55 is used to convert a single term into sort format. Given (code PWR base exponent NIL) and 1, SIMP55 creates ((((code·base) (code·exponent)))·(1·1)) which is the sort structure with one term, as it is passed up to SIMP5. In the case of PLS, the simplevel is not raised for the calculation of the exponent hash code number, as it is for PRD. SIMP54[X, Y, Z] converts a PLS or PRD expression X, to a sort list. Y is the group identity, and Z is T if X is a PRD expression. The conversion is straightforward, if a term has an exponent in the group sense, then it is used, otherwise 1 is used. The structure shown for SIMP55 is created. If the expression has a numerical coefficient, it is used rather than the identity. Returning to SIMP5, when all terms have been sorted, a result is passed up which depends both on the results of the sort and the level above. If there is only a numerical term, this is passed up. If the current operator is the same as the operator above, the sort format is passed up. Otherwise, the sort format is converted to standard form by SIMP53. If the expression part should reduce to a PLS when the current operation is PRD and the numerical part is not 1 and the operation above is PLS, then the distributive law is applied, the number is multiplied through the expression with a MAPL and the result is converted to sort format by SIMP54. Otherwise the number is just combined with the expression to produce the result to be passed up.

Returning to SIMP3, PWR produces a call to SIMP7. If the base is a PRD or PWR, then it may already be in sort format; if not, it is converted to sort format by a call to SIMP54 or SIMP55. If the base is not a PWR or PRD, the base and exponent are given to DPWR to produce the result. Otherwise, the sort format is given to SIMP71, which multiplies the exponent times each of the exponents on the sort format and simplifies each result with SIMP5. The number is exponentiated with DPWR. SIMP7 then checks to see how the result should be passed up. If the operation above is PRD or PWR, then the sort format is passed up. Since the result of DPWR applied to the number and the exponent may be an expression rather than a number, it is necessary to add the result to the sort list if it is not a number and take a dummy number of 1. If the above level is not PRD or PWR, then the sort format is converted to standard form by SIMP53 and then passed up.

Returning to SIMP3, an operation NIL indicates that the expression is an atom or FRT, and a hash code is generated by GETC and put on the front of the atom or FRT. Otherwise the expression is treated as an arbitrary function. The hash is computed by FGFN, and the hash numbers are removed from the arguments of the expression by SIMP31 before the operator and hash number are added to the front.

DRV and ITG are sent to SIMP8, which brings out a minus sign.

All expressions above the lowest level on the expression tree are sent from SIMP3 to SIMP6 where the list structure is altered as explained above.

SOLVE

SOLVE[EXP, X] applies the obvious transformations in order to solve an equation for a specified variable or expression, X. When all possibilities fail, it returns the equation in its partially solved state. SOLVE first checks to see if the expression is an equation and transfers to ERROR if it is not. Next, SOLVE substitutes the atom "SOLVE" for the expression X. This allows the use of MKP to compute dependence on X, no matter what the form of X. After the substitution, SOLVE1 calls EXPAND to multiply out both sides. SOLVE3 then sorts both sides into terms which depend on "SOLVE" and those which do not. SOLVE1 moves the dependent terms to the left side and the others to the right. The resulting terms are given to SOLVE2. The terms which depend on "SOLVE" are the first argument, those terms which do not are the second argument, and "SOLVE" is the third argument. SOLVE2 first checks to see if the dependent expression consists of only the variable itself, in which case the solution is accomplished; otherwise, it examines the main connective in order to apply a simplifying transformation.

If the connective is PLS it tries to factor a dependent term out of the left side, leaving only an independent term as a cofactor. The right side can then be divided by the cofactor. This is done as follows. SOLVE51 looks at the first term of the PLS, if it is a PRD, it checks to see if only one of the factors is dependent and returns this one. If more than one is dependent, SOLVE51 fails by returning NIL. If the first term of the PLS is not a PRD, SOLVE51 returns the entire term. SOLVE5 then tries to factor this dependent term out of each of the PLS terms, leaving an independent cofactor. It uses SOLVE52 to examine each term. SOLVE52 checks to see if the term to be factored out is the whole term, or if the term to be factored out is a factor of the whole term. In the case of a PRD it calls SOLVE521 to examine each factor. If SOLVE5 is successful, the left side is divided by the cofactor. Otherwise, the current state of the equation is the answer.

If the left side connective is PRD, SOLVE4 tries to remove exactly one dependent factor and divide the right side by the independent cofactor.

If the left side connective is PWR, one of two transformations is done. If the exponent is independent, the right hand side is symbolically taken to the converse of the exponent. Otherwise, the log of both sides is taken, the left side is given to SOLVE6 which multiplies it out and simplifies it and then removes any independent terms of the top level PLS to the right side.

If the main connective is NLOG, SIN, COS, TAN, ASIN, ACOS, or ATAN, then an inverse of the right side is taken. SOLVE21 checks to see if the left and right sides have the same one of these connectives, in which case it can just be removed from both sides.

SPLIT

SPLIT[EXP] makes two passes through the expression. First, SPLIT1 replaces the property list of each subexpression with a count of the number of subexpressions which that subexpression contains. Then, SPLIT2 uses these counts to reduce the total number of subexpressions in EXP to less than 100. It does this by naming parts of EXP and substituting these names for the parts. The named subexpressions are placed on the disk with EASS.

SPLIT1 uses several functions to count the subexpressions. It goes down to the bottom of the expression tree, and SPLIT3 computes the count on the way back up. To do this SPLIT3 adds the number of subexpressions in each of the arguments. It adds 1 to this sum in order to count the current subexpression. It adds 1 by considering the operator to be an argument. The number of subexpressions in a given argument is computed by SPLIT4. SPLIT4 counts an atom as 1; other arguments have their count in place of a property list.

SPLIT2 goes down the expression tree to prune it. If it reaches an atom, it leaves it alone, as it would save nothing to rename an atom. Also, if the expression size, M, is smaller than the allotted size N, it needs no pruning. Otherwise, the arguments are pruned in reverse order to their size. SPLIT6 finds the size R of the largest argument. If renaming only part of this argument will bring the total count below N, then SPLIT2 is applied to this argument with an allotment equal to the difference between N and the sum of the count for the other arguments. Otherwise, this argument has its count reduced to zero by SPLIT5 and SPLIT8, indicating that it is to be renamed. Then the R is subtracted from M, and SPLIT6 finds the next largest argument.

SUBSTITUTE

SUBSTITUTE[X Y Z] substitutes each member of set Y for the corresponding member of set Z in the expression X. It checks to see if Y and Z are sets and gives the arguments to SUBST1. The property lists are set to NIL so that expressions can be matched with EQUAL. SUBST1 gives the members of sets Y and Z to SUBST2, a pair at a time. SUBST2 substitutes the member of Y for each instance of the member of Z, with the following exceptions. a_i does not contain an instance of a. SUBST2 uses SUBST3 to avoid testing the a in a_i against the member of Z. Furthermore, SUBST2 will substitute:

A for B in (B X Y NIL) and

A_i for B in (B X Y NIL).

SUMEACH

If the argument of SUMEACH is of the form $\sum a + \cdots + b$, it will return $\sum a + \cdots + \sum b$.

SUMEXPAND

If the argument of SUMEXPAND is of the form $\sum_{i=n}^{m} X$, where m and n are integers, then SUMEXPAND will return the expanded summation.

TERM

TERM[X, N] returns the Nth argument of X or NIL if X has less than N arguments.

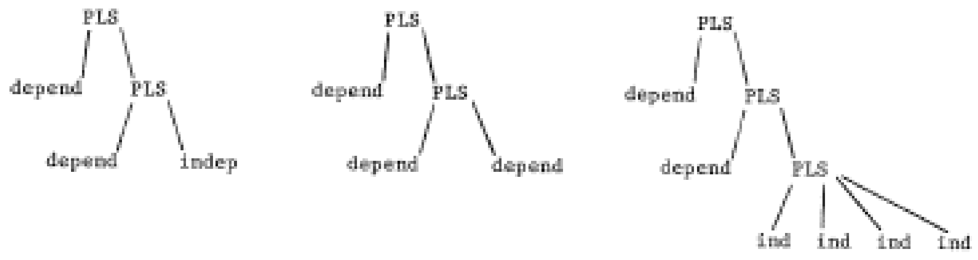
TRUNCATE

TRUNCATE[Y, X, N] finds terms to the N^{th} power in the expansion about the origin of Y as a function of X. Y must have a finite number of poles at the origin. Since Y is allowed to have poles at the point of expansion, it cannot be expanded by just evaluating successive derivatives at the origin. Furthermore, differentiation can often lead to very large expressions. Therefore, the method used here is to expand the smallest subexpressions in a power series, and then to combine these series to form the expansion of successively larger subexpressions. The initial expansions must contain enough terms so that the final combined result will be valid to power N. The required number of terms is calculated by making two passes through the expression. During the first pass, the minimum power of X in each subexpression is placed on the subexpression's property list. Then, the second pass uses these minimum powers as it goes down the expression tree in order to calculate the maximum required power of each subexpression expansion. The expansion of the individual atoms in a power series is simple, but many functions were needed to efficiently compute the expansion of each operator from the expansions of its arguments. The program is further complicated by the expansion of indexed summations. In only some cases can the program find the power of the variable of expansion as an increasing function of the index of summation. If this can be done and the summation has a finite lower limit, then the lowest power can be found. The second pass then carries out the expansion by repeated substitution. Arbitrary functions are expanded by differentiation. The arguments are still expanded in the normal manner before the differentiation, unless they also require differentiation. In this case, the differentiation is performed only once.

This routine is so large that it will not all fit in memory at one time. It has been split into two parts. The first part finds the minimum powers, associates a minimum and maximum power with each atom or constant subexpression and expands summations. The second part, which is compiled in the file TRNK9 LISP, combines the subexpression series in order to compute the final result. The most complicated part is the direct generation of given terms in a multinomial expansion. Each of the functions will now be described in detail. TRNK first defines MIN to have the NCODE function EMIN. It then defines MIN to have an NSIMP function which leaves the expression unchanged unless all of the arguments are numerical, in which case EMIN is used to find the minimum argument as a result. TRNK next calls TRNK11 to explore the expression, storing information on the expression's property lists. The expression is then passed to TRNK122 which chains to the second half of the program TRNK9.

TRNK11 passes each side of an EQN or all of any other expression to TRNK111, which applies TRNK7, then TRNK8, and finally TRNK21.

A <u>dependent</u> expression is one which contains the variable of expansion. TRNK7 applies TRNK71 to all dependent PLS or PRD subexpressions in the expressions. TRNK71 changes the PLS's or PRD's into a series of binary ones, such that any dependent argument is in a binary one, but the independent arguments are lumped together to produce one of the forms.



The independent terms are collected on the last argument of TRNK71. The function TRNK72 is used to avoid a redundant level if no independent terms have been collected.

The next step is to go down the expression tree in order to find the lowest power of the variable of expansion in each subexpression. The function TRNK8 goes down the tree to the individual variables and constant subexpressions. Coming back up the tree, the function TRNK82 dispatches control to the several functions which compute the lowest power of an expression from the lowest power of its arguments. The progress of TRNK8 down the expression tree is interrupted when a SUM subexpression is encountered. The lowest power of the variable of expansion may be expressed in terms of the variable of summation. TRNK8 calls TRNK83 to explore the expression. If the lower limit of summation is a number and the upper limit is a number or infinity, then the SUM body, B, expansion variable, X, and summation variable, i, are given to TRNK831. There are several cases where an expression for the lowest power can be found. If B is independent of X, then B is a constant and the lowest power is 0. If B is independent of i, then the lowest power can be found in the normal way, using TRNK8. Otherwise, the method used depends on the main connective of B. For PRD it is the sum of the lowest power of the arguments. For PLS it is the minimum lowest power of the arguments. PWR is handled only if it is of the form $f(X)^{g(i)}$; in this case the lowest power in f(X) is multiplied by g(i). If it is NAM, the lowest power is 0. Otherwise, the user is asked to return an expression for the lowest power as a function of i. The expression for the lowest power as a function of i which results from the above procedure is next checked to see that it is only a function of i and in addition an increasing one.

If these conditions are met, then the lowest power is found by plugging in the lower limit of summation and the exponent expression is saved by putting it on the property list of the SUM subexpression under the indicator SUM. Once again, if the exponent expression does not meet these conditions, the user is asked for the lowest power.

Coming back up the expression tree, TRNK82 dispatches control to the following functions. PLS and PRD are computed directly by taking the minimum and the sum respectively of the lowest powers of the arguments. PWR sends control to TRNK85. Only two cases can be handled; a zero exponent reduces the whole expression to 1, and an integer exponent can be multiplied by the lowest power of the base. In other cases the user is asked for the lowest power. DRV sends control to TRNK86. This function checks to see if differentiation is performed with respect to the variable of expansion. If so, the user must be asked for the lowest power. Otherwise, the lowest power is the same as that of the DRV body. ITG sends control to TRNK87. Here again, the user must be asked if the limits are dependent and otherwise the lowest power is the same as that of the ITG body. If the user is asked for the lowest power in DRV or ITG, then he must later be asked for the expansion. This is flagged by changing DRV and ITG to ASKDRV and ASKITG. All other connectives send control to TRNK88. They are treated as arbitrary functions which must be expanded by differentiation. Consequently, if any of the arguments have poles at zero, the user is asked, otherwise, the lowest power is taken as zero.

TRNK21 puts the highest needed power on the property list of the whole expression and gives the expression to TRNK2 which goes down the expression tree. When TRNK2 reaches an atom, it returns an ATOM structure with a minimum power of 1 or 0 depending on whether or not the atom is the variable of expansion. If the subexpression is independent, the maximum power is put on its property list and the downward recursion of TRNK2 is terminated. Otherwise, the maximum power needed for each of the arguments of the subexpression depends on its main connective. For PRD, the maximum power of each argument is the maximum power for the PRD minus the minimum power for the other argument. For PWR, it is now assumed that the exponent is an integer, the maximum power for the base is the difference between the maximum power for the PWR and (exponent-1)·(minimum power for base). SUM initiates a call to TRNK24. TRNK24 forms a PLS through expanding the SUM by substituting increasing integers starting with the lower limit of summation into the SUM body until the SUM is fully expanded or until the evaluation of the exponent function under the same substitution shows that the desired maximum power has been reached. The resulting PLS expression is then given to the top level function TRNK11 which applies both the minimum and maximum power passes. For all other main connectives, the maximum power for the arguments is that for the subexpression.

After TRNK2 has finished, the expression is given to TRNK122. This function creates a file LISPT LISP which contains the doublet TRNK123 (expression, max power, expansion variable), and then chains to the SAVED file LISPT. When control comes back, the answer is in file LISPTR LISP and this answer is read in and defined as a form to complete the operation of TRNK. The operation of file LISPT SAVED will now be described.

The top level function of LISPT SAVED, TRNK123, writes the file LISPTR LISP containing the result of applying TRNK12 to the expression generated by the main system. TRNK12 operates on each side of an EQN or the whole expression otherwise. TRNK9 is applied by TRNK12 to the expression and the resulting list of coefficients is given to TRNK121 which generates a PLS containing the terms of the desired expansion. TRNK121 just recurs down the list of coefficients using EPWR and EPRD, since the first coefficient is by definition that of the highest required power and no coefficients are omitted.

For purposes of efficiency TRNK9 uses a special data form, coming up the expression tree each function receives a list of coefficients, from the highest required to the start of all zero coefficients, with none omitted in between. TRNK9 goes down the expression tree to the individual variables and constant expressions, where it sets up the coefficient lists. It removes the ATOM level. TRNK51 is used to generate leading zeros for the coefficient lists. Downward recursion goes only through the body of ITG and DRV. For ASKITG and ASKDRV the user is asked for the expansion. For PWR, two cases are distinguished. If the exponent times the lowest power of the base is greater than the highest power required, then the coefficient list is NIL. Otherwise, TRNK9 is applied only to the base. In the case of NAM or an arbitrary function, a check is made to see if any of the arguments have poles. If they do, then an effort is made to remove them by applying TRNK9 to the arguments and then converting the result back to normal form. The third argument of TRNK9 is set as a flag, for the NAM or arbitrary function subexpression must be expanded by differentiation, and this need not be repeated if the same situation is encountered at a level within the subexpression. If the arguments do not have poles at zero, the differentiation step is applied to the subexpression once the atom structure has been removed from it by ATOMOFF.

Coming back up the expression tree, the coefficient lists are given to functions associated with the main connective at the particular node. PLS sends control to TRNK1. TRNK1 adds corresponding coefficients with EPLS until one of the two coefficient lists is exhausted, the remaining coefficients of the other list are then taken as is. PRD sends control to TRNK5. If either argument list is NIL, all coefficients are zero on that list so the result is NIL. Otherwise, the arguments are passed to TRNK2222 in a special form best illustrated: $(0\ 0\ 0\ a_0\ a_1\ a_2)$

$(b_3\ b_2\ b_1\ b_0)$

The number of leading zeros in the first list is one less than the number of coefficients in the second list, and the coefficients in the first list have been reversed. TRNK2222 removes the zeros from the first list, one by one. Before removing each zero it sends both lists to TRNK3, which generates one coefficient in the answer. This coefficient is a sum of terms generated by multiplying by means of EPRD corresponding terms in the above lists. This TRNK2222 process is terminated by a final call to TRNK3 once all zeros have been removed.

For PWR, there are two possible calls. If the exponent is positive, SRPWR is called and TRNK10 is then used to remove trailing 0's from the coefficient list which SRPWR produces. SRPWR takes four arguments, the coefficient list, the maximum power on this list, the maximum power needed in the result, and the exponent to which the series represented by the coefficient list is to be raised. If the coefficient list contains coefficients of negative powers, the SRPWR2 is called; otherwise, SRPWR3 is called. The operation of SRPWR3 will be described first. The purpose of this function is to generate the coefficients of the desired terms in the multinomial expansion, without generating the entire expansion. For each term, SRPWR3 initiates a call to SRPWR1. The desired coefficient could be written as a sum of all products of the coefficients of the base series which have the form

$$N! \frac{a_0^{i_0}}{i_0!} \frac{a_1^{i_1}}{i_1!} \frac{a_2^{i_2}}{i_2!} \cdots \frac{a_n^{i_n}}{i_n!} ,$$

where $i_1 + i_2 \ldots + i_n = N$, the exponent, and $1 \cdot i_1 + 2 \cdot i_2 \ldots n \cdot i_n = P$, the desired power. The desired coefficient is, however, in fact, written with the above sum factored on the a_j's, taken in the dictionary order. The coefficient is built up in this manner by an exchange of calls by SRPWR1 and SRPWR11. SRPWR1 has as arguments the needed power, P, the number of coefficients yet to be used M, the power R, which the next coefficient in the dictionary order can be used to generate, the power, V, of this coefficient in the base series, the minimum power, S2, in the base series, and the list, A, of remaining coefficients. SRPWR1 recurs on the argument R. R is started off at the minimum value which will allow generation of a term, it is then increased by 1 until $R \geq M$, in which case more the N total base coefficients would be used, or until $P-(R+1)*V < S2*(M-(R+1))$ which states that if the remaining M-(R+1) powers were all used on the lowest coefficient S2, the total power generated would still be greater than that required. The terms created for each value of R are added with EPLS. Each of these terms is created by a call to SRPWR11. SRPWR11 checks to see if the coefficient list, A, has only one more member. In this case it creates the desired term $(\text{coef})^R/R!$; otherwise, it constructs this term, but multiplies it by the term created by giving the remaining coefficient list to SRPWR1, along with the remaining number of powers needed, P-RV, the remaining coefficients available, M-R, and the lowest value of R which you can start with next, MAX[0, P-2R+M·(2-V)]. The expression P-2R+M·(2-V) is best viewed as (P-RV)-(M-R*(V-2); P-RV, as given above, is the number of powers still needed, (M-R)*(V-2) is the number of powers which would be produced if none of the next coefficient were used. The difference is the number of powers which must be used on the next coefficient, for using a coefficient on the next, rather than the one after next, will increase the generated power by one.

SRPWR2 has six arguments. N is the maximum power needed, X is the list of positive coefficients, Y is the list of negative coefficients in ascending order. S2 is the lowest negative coefficient and S1 is the highest positive coefficient. S is the exponent to which the base series is raised. SRPWR2 uses four program variables: J, U, M, and P. The result is built up on U and is a sum of terms for each coefficient. The current coefficient is J, which is indexed from the lowest which can be made up to N, the highest needed. For each value of J, a coefficient is built up which is a sum of terms. Each of these terms is of course a product of negative and positive coefficients such that the sum of the powers is J, and the number of coefficients is S. The terms can be classified by the number, M, of negative coefficients and the number, P, of negative powers which they contain. SRPWR generates all possible values of M and P which will produce the given J; it then calls SRPWR1 for each pair of values of M and P, once to produce the negative part of the coefficient, and once to produce the positive part. The positive and negative parts are then multiplied together with EPRD. The values of M and P are generated as follows. M is started off at 0 if J is positive, or so that M*J $\leq$ S2 if J is negative. M is then incremented until M $>$ S1*(S-M)-J. The meaning of this relation is as follows. Since the highest value of negative base coefficient is -1, M negative coefficients must create at least M negative powers. The maximum number of positive powers which can be generated with the remaining S-M coefficients is S1*(S-M), but J of these are needed for the result, so that S1*(S-M)-J can be matched against negative ones and this number must be greater than or equal to M. Now, for each feasible number of negative coefficients, terms for corresponding possible negative powers are generated. The number of negative powers is given by P, which is started off at the minimum of -M*S2, the most negative you can make, and S1*(S-M)-J, the most negative you can absorb with extra positive powers. P is then decremented until it is less than M or until it is less than -J, which means that if J is negative, not enough negative powers to produce it will be generated. Each completed coefficient is multiplied by S!.

Returning to TRNK9, if the exponent of PWR is negative, then a call to TRNK63 is given. TRNK63 inverts the series created by SRPWR as for a positive exponent by a call to TRNK6. TRNK6 first checks for a null series which would indicate division by zero is called for; in this case it notifies the user. Otherwise, it sets up TRNK61, which does the inversion. TRNK61 builds up the answer, term by term, on its first argument W. It recurs on its last argument N which is a term counter, incremented from 1 up to the number of coefficients on the list U, of the series to be inverted. For each value of N a term computed by TRNK62 is added to W. This term is $-\frac{1}{a_0}$ times the sum of the product of corresponding terms of W and U.

Returning once again to TRNK9, ITG and DRV send control to TRNK93. Since only the case where the integral or derivative do not involve the variable of expansion is handled, TRNK93 goes through the coefficient list putting the integral or derivative structure and arguments on each coefficient. The only remaining cases for TRNK9 are NAM and an arbitrary function; they are handled in the same manner. The arguments are treated as explained on the downward pass and then the expression in standard form is passed to TRNK95 for expansion by differentiation. It is at this point that the check is made to see if the subexpression under consideration is contained in a larger one which must also be expanded by differentiation, in which case the expression is returned undifferentiated by TRNK95. Otherwise, TRNK95 calls TRNK13 to expand the expression by differentiation. TRNK13 uses DIFF2 to repeatedly differentiate the expression, which it keeps on program variable U. Before each differentiation a term of the expansion is set up by substituting zero for the variable of expansion in the expression U, using TRNK131. TRNK131 substitutes zero for the variable of expansion, except that if it comes to a DRV subexpression it creates an EVL level.

This completes the explanation of TRNK.

Chapter VII

HASH CODING FUNCTIONS OF A COMPLEX VARIABLE

Introduction

Several of the routines described in the preceding chapter require the comparison of algebraic expressions for equivalence. The comparison is made by hash coding the expressions and comparing the resulting hash code numbers. The formulas required for the hash coding scheme and the functions which apply them were described in the last chapter. The mathematical considerations underlying these formulas will now be described.

The elementary functions of a complex variable are those which can be expressed by the following recursive scheme. Any complex constant or variable will be called an expression; if u and v are expressions, then so are $u + v$, $u \cdot v$, u^v, e^v, $-u$ and $\frac{1}{u}$. The trigonometric and hyperbolic functions may be expressed explicitly. Because of the defining relations of the complex field and the trigonometric identities, there are infinitely many expressions for any given function. Two expressions will be said to be equivalent if they represent the same function.

Existing schemes for expression comparison use the defining relations along with some additional conditions to put each expression in a canonical form. If the canonical forms of two expressions are identical, they must represent the same function. This method has certain drawbacks. First, putting the expression in a canonical form requires the comparison of many subparts of the expression with each other. In particular, the commutative law requires that the terms in sums and products be sorted. Second, discovery that two expressions are equivalent requires a comparison of every subpart of one with the corresponding subpart of the other. Third, the problem of reducing all equivalent expressions to one canonical form is recursively unsolvable (7) and the existing schemes fail in many cases.

This chapter explores a probabilistic approach. Suppose $F(z) \not\equiv G(z)$ (F and G are elementary functions), then $F(z) - G(z) = 0$ has, at most, a countable number of solutions, while the complex numbers are uncountable. Therefore, the probability that $F(z) - G(z) = 0$ for a point z chosen at random is 0. Thus, it would be possible to test for equivalence of expressions by comparing their values at a randomly selected point. It is possible to get some approximation to this fact with the finite arithmetic of a computer.

One method would be to substitute a random floating point number for each occurrence of each distinct variable and then evaluate the resulting expression using floating point arithmetic. This method is limited by overflow and roundoff error. For example, if x is a floating point number chosen at random from a flat distribution, then with probability one half x^2 is larger or smaller than all the floating point numbers; it does not appear possible to find a rule for mapping x^2 back into the floating point numbers such that the code numbers of equivalent expressions will be very nearly the same. This overflow is difficult to avoid by restricting the initial choice of floating point numbers since expressions of the form u^v are allowed. Furthermore, if two expressions, x and y are of different orders of magnitude, then, because of roundoff error, $x + y$ may evaluate to either x or y. This is a particularly serious disadvantage since it is likely that an expression will be compared with subparts of itself. The same problems arise with a floating point approximation to the complex numbers.

Another strategy, which we investigate here, is to use a finite field, instead of the infinite field of real numbers.

Finite Fields and the Exponent Arithmetic

a. Finite Fields

The use of floating point numbers in the code number scheme is limited because the sum or product of two floating point numbers is not necessarily a floating point number. This problem is avoided if a finite field, F, is used, since the field can be chosen small enough so that every element can be represented by a computer number. The task is to choose F such that expressions which are equivalent in the complex numbers are also equivalent in it. That is, we need a homomorphism from the complex numbers onto F. We now develop a field which meets this requirement in many, but not all, cases.

An abelian group G is a set of elements with an operation x and an identity element e such that:

1. $a \in G$, $b \in G$ then $a \times b \in G$
2. $a \in G$ then $ae = ea = a$
3. $a \in G$ then $\exists\, a^{-1} \in G \ni aa^{-1} = a^{-1}a = e$

4. $a \in G$, $b \in G$ then $ab = ba$

A finite field F is a finite set of elements with an operation + under which the elements of F form a group with identity 0 (the additive group), and an operation $*$ under which the elements of $F' = F - 0$ form a group with identity 1 (the multiplicative group). In addition the relations

$a^* (b + c) = a^*b + a^*c$ and $a^*0 = 0$

hold.

If m and n are integers and p is a prime integer then $c = (m + n) \mod p$ means that c equals the remainder of $(m + n)/n$. Multiplication mod p is defined similarly. It can be verified that the integers less than a prime form a finite field under the operations addition and multiplication mod p. The additive inverse of 1, -1 is seen to be $p - 1$ since $p - 1 + 1 = 0 \mod p$.

b. <u>The element i</u>

In the complex field there is an element i such that $i^*i = -1$, so such an element is also required in F. To see how this restricts the choice of p one needs the fact found in the references that the multiplicative group F' of a finite field is cyclic. This means that there is an element α (called a generator) in F' such that every element in F' is some power of α. In fact, F' can be written $1, \alpha, \alpha^2, \ldots, \alpha^{p-2}$ and $\alpha^{p-1} = 1$. Since p is a prime it is odd and so $\frac{p-1}{2}$ is an integer. $\alpha^{(p-1)/2} = -1$ since $\alpha^{(p-1)/2} \neq 1$ and $(\alpha^{(p-1/2)})^2 = 1$. If $\frac{p-1}{2}$ is even then $r = \frac{p-1}{4}$ is an integer such that if $i = \alpha^r$, $i^2 = -1$. Note that either α^r or α^{3r} can be chosen as i and the other becomes $-i$. We have thus shown that F will have an element i if and only if p is of the form $4q + 1$.

c. The Exponent Arithmetic

In the complex numbers, one might have to test for the equivalence of two expressions such as u^{v+1} and $u^{v} \cdot u$, where the exponent arithmetic is also performed in the complex numbers. However, since the multiplicative group is a cyclic group with one less element than F, $v + 1$ must be computed mod $(p - 1)$. Since an isomorphism does not exist between the additive and multiplicative groups of F, the exponent operations cannot be performed in it. This failure of the finite field evaluation to be recursive in the exponent direction is a serious limitation. Furthermore, since $p - 1$ is not a prime the exponent operations will not form a field. Fortunately, many expressions encountered in analysis have rather simple exponents and so much can be saved by evaluating the exponents in the E arithmetic which we now define.

Let the basic elements of E be the integers less than $p - 1$. Addition and multiplication are mod $(p - 1)$. It is easy to see that all elements have an additive inverse. No even integers have a multiplicative inverse, however, for this would imply:

$$2 \cdot s \cdot (2 \cdot s)^{-1} = 4 \cdot q \cdot m + 1$$

or

$$2 \cdot (s \cdot (2 \cdot s)^{-1} - 2 \cdot q \cdot m) = 1$$

which is a contradiction since 1 has no divisors in the integers. If we take q prime, then the odd integers other than q have a multiplicative inverse as a consequence of the Euler theorem (see Ref., Albert, p. 47):

Let ϕ (m) be the number of integers g such that $0 < g \leq m$ and g is prime to m. Then

$$a^{\phi(m)} = 1 \bmod m$$

for every a relatively prime to m.

The failure of the even integers to have a multiplicative inverse means that $u^{1/2 \ + 1/2}$ will not evaluate to u. We therefore adjoin to the basic elements of E the element ϱ, where $2\varrho = 1$. Closing E under multiplication and addition would require that all the elements of the form $b\varrho^i$ (b an integer less than p - 1 and b odd) be in E. However, many cases can be covered if we allow only elements a or $b\varrho$.

d. Square Roots in F

u^{ϱ} should evaluate to be the square root of u in F, however, only one half the elements in F have a square root, these are the even powers of the generator, α, of F'. Since $(\alpha^n)^r$ is even for even n and any r only $\frac{1}{4}$ of the elements could have a square root computable by raising the element to some power, that is by assigning some integer to ϱ. However a method of finding the roots of this smaller set can be found, as will be shown next, i.e., there exists a ϱ such that if $u = \alpha^{4n}$, then $u^{\varrho} = \alpha^{2n}$.

If p is of the form 4q + 3, then since $4n(q + 1) = 2n \bmod (4q + 2)$ for $n < q$ one finds $(\alpha^{4n})^{q+1} = \alpha^{2n}$. q + 1 is therefore the proper value for ϱ. The requirement that p be of the form 4q + 3 is unfortunately in conflict with the earlier requirement that p be of the form 4q + 1. By choosing $p = 8q' + 5 = 4(2q' + 1) + 1$ one obtains by a similar argument the square root of 1/8 of the elements with $\varrho = q' + 1$.

e. Trigonometric Identities

Define in the usual manner:

$$\sin\theta = \frac{e^{i\theta} - e^{i\theta}}{2i} \qquad \sinh\theta = \frac{e^{\theta} - e^{-\theta}}{2}$$

$$\cos\theta = \frac{e^{i\theta} + e^{-i\theta}}{2} \qquad \cosh\theta = \frac{e^{\theta} + e^{-\theta}}{2}$$

where e is an element of F yet to be chosen. Note that $i^2 \neq -1$ in the E arithmetic but this does not arise in taking sums and products of the above functions. For instance:

$$\sin^2\theta + \cos^2\theta = \left(\frac{e^{i\theta} - e^{-i\theta}}{2i}\right)^2 + \left(\frac{e^{i\theta} + e^{-i\theta}}{2}\right)^2$$

$$= \frac{e^{i\theta}\cdot e^{i\theta} - 2\cdot e^{i\theta}\cdot e^{-i\theta} + e^{-i\theta}\cdot e^{-i\theta}}{-4} + \frac{e^{i\theta}\cdot e^{i\theta} + 2e^{i\theta}\cdot e^{i\theta} + e^{-i\theta}\cdot e^{-i\theta}}{4}$$

$$= \frac{-2\cdot 1}{-4} + \frac{2\cdot 1}{4}$$

$$= 1$$

If $\sin(\theta)$ is to equal $\cos(\pi/2 - \theta)$, then it is necessary that $e^{i\pi} = -1$ and $e^{i\pi/2} = i$. If e is to have a square root it must be an even power of α; taking $e = \alpha^{2n}$ one obtains for $p = 8q' + 5$:

$$e^{i\pi} = -1$$

$$\alpha^{2ni\pi} = \alpha^{4q'+2}$$

$$2ni\pi = (4q' + 2) \bmod (8q' + 4)$$

$$ni\pi = (2q' + 1) \bmod (4q' + 2) \qquad 1)$$

For any choice of n, e is determined, providing equation (1) can be solved for the element π. Some reflection will show that trigonometric calculations may involve roots of e greater than 2. Suppose n is chosen odd, then one square root of e, α^n, has no square root, nor does $-\alpha^n$, the other square root of e. That $-\alpha^n$ does not have a square root is a consequence of the choice of -1 as a square. $-\alpha^n = -1 \cdot \alpha^n$ = (square) (nonsquare) and a square times a nonsquare must be a nonsquare. From this one can see that if e is to have a 2^mth root, it must be chosen of the form $\alpha^{c \cdot 2^m}$ Note that the choice of n divides the elements of F' between the roots and powers of e.

Returning to the solution of equation (1) for the case n = 1, we see that no satisfactory solution is possible. For (1) implies that $i\pi = 2 \cdot m \cdot (2q' + 1) + (2q' + 1)$. $2q' + 1$ must be taken prime in order to reduce the number of elements which map into the same element in the E arithematic. Therefore, $2q' + 1$ must divide either i or π, but this is not acceptable. Suppose $2q' + 1$ divides π, then in the E arithematic $4\pi = 0$. Since either 4π or $4i$ is a highly probable element, this relation is unacceptable. Once again, a patch can be inserted so that $e^{i\pi/2}$ will equal i in the most frequent cases. When an element is exponentiated in the F arithematic, special checks are made. If the base is e and $i\pi$ divides the exponent, then $(-1)^{\exp/i\pi}$ is computed. $i\pi$ is chosen small enough so that small multiples of it will be less than the prime p.

f. Summary of the Requirements

1. p of the form 8q' + 5.
2. 2q' + 1 prime

Machine Realization -- Finding a Prime

The requirements of section II are:

1. p = 8q' + 5
2. 2q' + 2 prime

Another requirement is:

3. p less than 1/2 the largest machine integer.

This allows addition without overflow. The multiplicative inverse of an element a in F is found by noting that $a^{-1} = a^{p-2}$. To raise an element a to any power we begin by multiplying it by itself, creating the numbers $b_i = a^2$, we then express the power as a binary number and add up the appropriate b_i. This leads to the requirement:

4. p - 2 should be expressible as a few powers of 2.

To find a prime for the 7094 the following procedure was followed:

1. Beginning with $n = 2^{29}$ test if p = 16n + 13 is prime by dividing by every odd number up to $\sqrt{p}$.
2. Test if 4n + 3 is prime.
3. Find a generator α, of F'. (If a is not a generator then $a^2=1$ or $a^4 = 1$ or $a^{\pi} = 1$ or $a^{2\pi} = 1$. Raise a to these four powers by the scheme above.) Almost 1/2 of the elements are generators so one is quickly found.

4. Compute $i = \alpha^{4n+3}$

This procedure resulted in:

p = 8, 589, 949, 373

α = 13, 560, 097

i = 5, 525, 736, 173

Some Example Problems

Ten trigonometry identity problems were selected from a textbook (8). The scheme produced the same hash number for both sides in all but the last. The identities are:

1. $\sin x \tan x + \cos x = \sec x$
2. $(\sin x \cot x + \cos x)/\cot x = 2 \sin x$
3. $\csc^2 x + \cot^2 x + 1 = 2/\sin^2 x$
4. $\cos x \cot x + \sin x = \csc x$
5. $(1 - \sin x)(\sec x + \tan x) = \cos x$
6. $\sin x/(1 - \cos x) = \tan x/(\sec x - 1)$
7. $\csc^4 x - \cot^4 x = \csc^2 x + \cot^2 x$
8. $(\sin x/(\sec x + 1)) + (\sin x/(\sec x - 1)) = 2 \cot x$
9. $\cos^6 x + \sin^6 x = 1 - 3 \sin^2 x \cos^2 x$
10. $\sqrt{(\sec x - 1)/(\sec x + 1)} = (1 - \cos x)/\sin x$

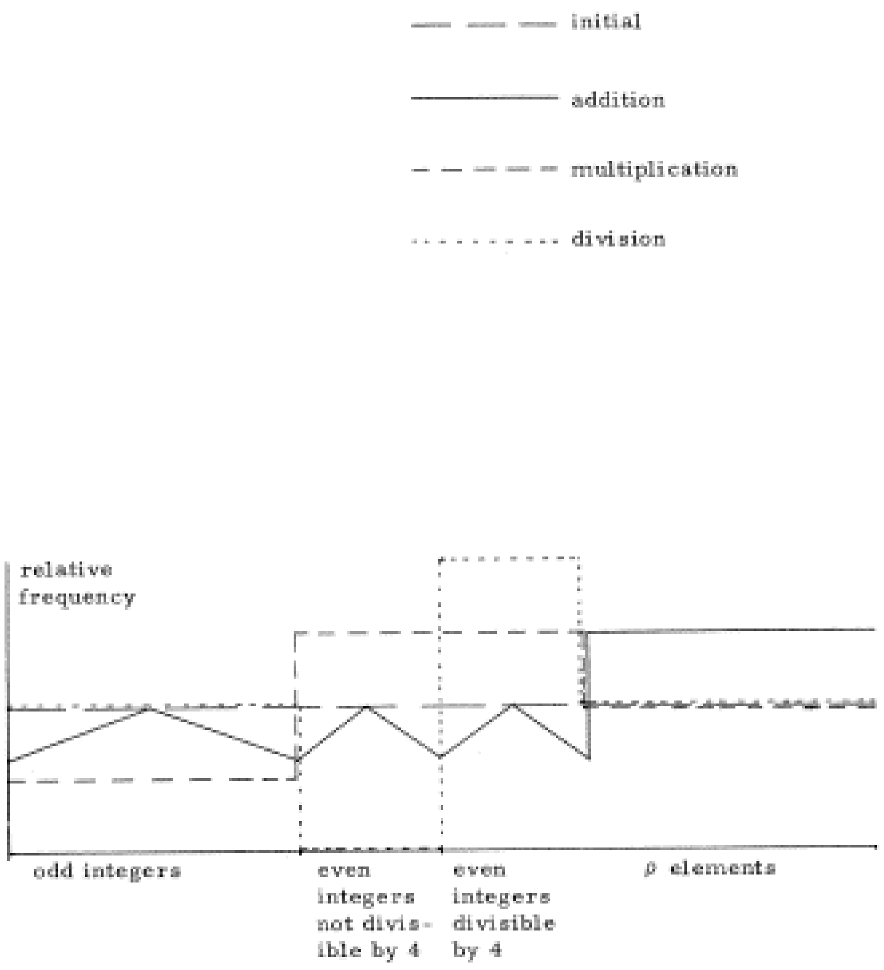
The probability of Error

Estimation of the probability of error is difficult. The average probability of error for certain subsets of expressions will differ from that for all expressions. No statistics are available on the expressions which will be encountered in practice.

It is possible that two expressions which represent the same function will receive different code numbers because some exponent operation does not preserve the equivalence. Study of section II should make clear under what circumstances this will happen.

In the simplification program described in the Chapter VI , expressions with the same code number are considered equivalent. Therefore, an accidental match of non-equivalent expressions is very serious. If we could show that the operations in the E and F arithmetic mapped their sets of elements uniformly back onto themselves, then the probability of a match between two expressions selected at random from the set of all expressions would be $1/p$. Unfortunately, this is not the case. In the F arithmetic the operations of multiplication and addition and their inverses do satisfy this criterion. Looking at the cyclic group F', however, one sees that raising any element in F except a multiple of π to all powers will produce either $1/4$, $1/2$, or all the elements of F. Thus exponentiation tends to map the elements into the 4th powers of a generator and so increase the probability of random match.

The same bunching occurs in the E arithmetic. The distribution of elements after n operations can only be found using a rather complicated two dimensional convolution. The distributions after one operation shown in Figure 1 indicate that for moderately complicated exponents the probability of error should remain in control.



Distribution of elements in E after one operation. Within each classification, the elements are ordered in the normal manner.

Figure 1

Chapter VIII

THE LANGUAGE OF MATHEMATICAL EXPRESSIONS

The introduction stated that man-computer symbiosis can bring about an effective problem solving team by assigning to man and computer those parts of a problem which they are best suited to solve. Each is much better than the other at certain jobs; therefore, the two together can do the problem better than either alone, if the cost of breaking the problem up is not too high. The task assignment may result in either parallel or serial operation of man and computer. In a parallel operation, each has continuous access to all important data generated by the other, while in a serial operation, data exchanges occur only at significant points in the problem. Sometimes, a task in a serial operation could be performed by either man or computer; the one which has performed the previous step often has the relevant information for the task. The savings in sending the task to the best partner must be great enough to cover the cost of a possible data exchange. In this case, the cost of data exchange is critical to the balance between man and machine. This would not be so if most tasks could only be performed by one or the other.

The solved problems presented in Chapter II require some serial operation with much data exchange. It is, therefore, important to investigate the possibilities for man-machine communication in non-numerical analysis. A language consists of a series of statements following certain syntatic rules which the communicants map into their respective models of the world.

Often, the language can be divided into a central part, following a rather small set of well defined rules, together with a series of succeedingly complex exceptions to these rules and methods for continued discourse. In the case at hand, the central part is the set of mathematical expressions used in analysis. These expressions will be investigated in this chapter and the following two chapters.

The available input-output devices for the language are the typewriter and the display. Either a one-dimensional or a two-dimensional mathematical expression syntax can be used with the typewriter or the display. The use of a one dimensional syntax for typewriter input is discussed in chapter X. Chapter IX describes a two dimensional syntax for display output. A two dimensional syntax for the typewriter is discussed by Klerer, however, he has not stated any useful general ideas. The problems involved in constructing a two dimensional input syntax for the display will be considered in Chapter XI.

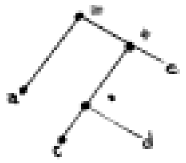
Chapter IX

SYNTAX AND DISPLAY OF MATHEMATICAL EXPRESSIONS

Introduction

This chapter describes a LISP program which computes the pictorial representation of a mathematical expression from a LISP S-expression representation of the expression. Since the language of mathematics contains a number of special cases, the computation will be introduced with a summary of the results produced by each part of the program and the general principles followed.

Because of the possible use of many transformations and identities, a mathematical expression can be represented in many equivalent forms. For instance a (b+c) could be represented as ab + ac. In certain contexts the picture of one of these forms may be easier to read than the picture of the others. For instance a - x might be preferred to a + (-1 · x). On the other hand, the use of several equivalent internal forms complicates the writing of routines needed to transform mathematical expressions. Furthermore, the most convenient internal form might be difficult to read. In the first pass of the picture compiler the internal form is transformed into an equivalent form which is easy to read. In addition, parentheses are added. The LISP expression resulting from this first pass has the form of a tree of subexpressions. Each node of the tree is a mathematical operation and the branches from that node are the arguments of that operation. At the end of the branches are the individual variables and constants. The form of this internal tree is preserved throughout the remaining steps of the picture compilation. For example, the expression $a = c \cdot d + e$ has the tree:



Mathematical operations are represented in "pictures" by symbols placed in special arrangements. A classification of the representations will be given later. The situation is complicated by the fact that sometimes the same symbols and positional notations are used in different combinations to represent different mathematical operations. The second pass of the picture compiler rewrites the internal tree in terms of these common symbols and positions. The symbols and positions are expressed in terms of a set of special forms. These forms are shown in Figure 1 through Figure 11. The compiler starts at the base of the internal tree and works out to the ends of the branches, rewriting the tree at each node. This rewriting introduces additional nodes into the internal tree, but these new nodes are marked so that the old structure is preserved. The display resulting from a node and its branches is called a picture part. Sometimes the form of a picture part depends on the dimensions of its arguments. In this case the second and third pass must be applied to the arguments before the second pass can be applied to this part.

In the third pass each picture part is inscribed in a rectangle. The compiler starts at the ends of the branches of the internal tree and works toward the base. The compiler first inscribes each individual constant and variable in a rectangle, defined by its width, height, and depth. A dimensioned rectangle is then chosen in turn for each larger picture part; i.e. operator with arguments. The dimensions of each rectangle depend on its own operator and also on the dimensions of those rectangles associated with the arguments of the picture part. In addition, we compute the relative positions of the arguments of each picture part with respect to the lower left hand corner of its circumscribed rectangle.

Once the entire expression has been _dimensioned_, it can be positioned on the oscilloscope face. The final pass of the compiler then sends to the display the name, size, and position of each symbol. These symbols are interspersed with non-displaying left and right pseudo-parentheses. These pseudo-parentheses group the picture symbols into a tree of sub-expressions identical with the internal tree structure obtained from pass one of the picture compiler. _Thus, a light-pen reference to a picture symbol can be identified with the smallest subexpression in the picture tree which contains it, and with the corresponding LISP subexpression in the internal tree_. This makes it possible easily to designate mathematically meaningful picture-parts as arguments for other operations.

Transformations to Facilitate Semantic Interpretation

A method is needed for referencing meaningful subparts of the LISP source expression by pointing with a light pen to symbols in the displayed picture expression. Thus, before the picture is compiled, the source expression is transformed using mathematical relations into a form which makes it easy to establish a one-to-one mapping between subparts of the picture expression and mathematically meaningful subparts of the LISP source expression. Then, in addition to the picture symbols, the compiler sends the display a series of left and right pseudo-parentheses which parse the picture symbols into a tree identical with the tree formed by the transformed source expression. When the light pen is pointed at a picture symbol, the smallest picture sub-expression which contains it is intensified and the corresponding LISP expression may then be referenced. The source expression is transformed in order to bring it into a better pictorial form, for instance, the order of the product is reversed so that $(\int_a^b cdx)\cdot d$ can be written $d\cdot\int_a^b cdx$.

Although the picture compiler is powerful enough to handle most all of the notations which arise in mathematics, simple compiler rules have only been written for the mathematical expressions listed below, which arise in the solution of non-linear differential equations. These expressions are represented by LISP S-expressions. In brief, S-expressions are defined recursively as follows: Any number or string of alphanumeric characters is an S-expression. One or more S-expressions separated by spaces and surrounded by parentheses is an S-expression. Individual numbers and alphanumeric strings are called <u>atoms.</u> Individual variables and constants in the mathematical expressions are represented by atoms. The mathematical operators are represented by S-expressions consisting of the operator name, its arguments in some given order, and the atom NIL. NIL denotes the null S-expression; it is replaced by temporary results during the transformation of expressions and represents the <u>property list</u> of the mathematical operator. The individual mathematical operators currently in the program are:

(Note: most of the operators can take any number of arguments, in the obvious manner.)

1. (PLS A B C NIL) $\equiv$ $A + B + C$

2. (PRD A B C NIL) $\equiv$ $A \cdot B \cdot C$

3. (FRT I J NIL) $\equiv$ $\frac{I}{J}$

4. (PWR A B NIL) $\equiv$ A^B

5. (DRV A B C D E NIL) $\equiv$ $\frac{d^{B+D}}{dA^B dC^D} \quad E$

6. (ITG D A B C NIL) $\equiv$ $\int_A^B \quad CdD$

7. (SUM A B C D NIL) $\equiv$ $\sum_{A=B}^{C} \quad D$

8. (EVL A B C D E NIL) $\equiv$ $E\Big|_{A=B}^{C=D}$

9. (NAM A B C NIL) $\equiv$ $C_{A,B}$

10. (F A B NIL) $\equiv$ $F(A, B)$

11. (NAM A B (F C D NIL) NIL) $\equiv$ $F_{A,B}(C, D)$

12. (FTL A NIL) $\equiv$ $A!$

13. (ABS A NIL) $\equiv$ $|A|$

Where A, B, C, D and E are arbitrary expressions, I and J are integers, and F is any atomic symbol not recognized as an operator. -X is represented by (PRD -1 X NIL) and $\frac{1}{X}$ is represented by (PWR X -1 NIL). If one of the factors of a sum or product is a number, it is often the left-most argument.

The first compiler pass performs the following transformations:

(PLS A NIL) → A

(PRD A NIL) → A

(PLS N A B NIL) → (PLS A B N NIL)

(DRV S 1 (F S NIL)NIL) → (DRV 1 1 (F S NIL) NIL)

-N → (NEG N NIL)

(PRD -1 A NIL) → (NEG A NIL)

(PRD -2 A NIL)→(NEG(PRD 2 A NIL) NIL)

(PRD A (SUM A B C D NIL) E NIL)→ (PRD A E (SUM A B C D NIL)NIL)

(PRD A (DEL B NIL) C NIL) → (PRD A C (DEL B NIL) NIL)

(PRD A (PWR B -1 NIL)NIL) → (DVD A B NIL)

(PRD A (PWR B (PRD -1 C NIL)NIL)NIL) → (DVD A (PWR B C NIL)NIL)

(F A B NIL) → (FUNCTION F A B NIL)

(PRD A (I TG A B C D NIL) E NIL)→(PRD A E (ITG A B C D NIL)NIL)

(FRT 1 -2 NIL)→ (NEG (FRT 1 2 NIL)NIL)

X → (ATOM X NIL)

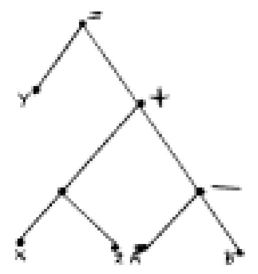
(NAM X Y (F A B NIL)NIL) →(FUNCTION (NAM X Y F NIL) A B NIL)

Where N is a number, X is a number or variable symbol, and S is a non-atomic subexpression.

In addition to these transformations, levels of parentheses are inserted. For instance (PRD A (PLS B C NIL)NIL) → (PRD A (PAREN(PLS B C NIL)NIL)NIL). No parentheses are used around the arguments of FUNCTION, PAREN, ABS, EVL or NAM. A size change always removes the need for parentheses. In addition, the integration and summation symbols stand in place of parentheses for operators on their left and a transcendental function is parenthesized if it is raised to a power. Otherwise, parentheses are placed around an argument of an operator if the main operator of the argument has lower precedence. Precedence is determined from the following list: PAREN, NAM, =, FUNCTION, FTL, PWR, FRT, PRD, =, DVD, ITG, DRV, NEG, =, PLS, =, SUM. Operators separated by = have equal precedence.

The expression resulting from these transformations is saved as the internal tree for the later analysis of light pen references. For example, the expression $Y - X^2 + \frac{A}{B}$ would be represented by the S-expression (EQN (ATOM Y NIL) (PLS(PWR (ATOM X NIL) (ATOM 2 NIL)NIL)

(DVD (ATOM A NIL) (ATOM B NIL) NIL)NIL)NIL). It has the tree structure



Transformations to Facilitate Syntactic Construction

To a very large extent, mathematical notation becomes, in the mind of the individual, a model of the abstract concepts he is manipulating. For instance, consider the operation of canceling terms from both sides of an equation or from the numerator and denominator of an indicated division, or the operation of bringing a term to the other side of an equation, or the operation of matrix multiplication. Therefore, the notation must have properties which make it easy to visualize.

Lack of generality is thus introduced in the course of making mathematical notation easier to read. Different mathematical operators should be represented by different sorts of notation and the mathematical operator with the highest precedence should be represented by the most compact form. For example, compare the equivalent Boolean expressions $(((A \cup B) \cup C) \cup (D \cap E))$ and $A + B + C + DE$. Furthermore, when an operation is applied to complex arguments, it should be possible to visualize it in the same manner as when it is applied to simple ones. For instance, the introduction of parentheses or other such <u>fences</u> allows $(A + B)C$ to be visualized in the same manner as DC, with the operation expressed as concatenation. Sometimes a change of size is used to aid in visualizing a complex argument as a single unit. As a final example, consider the choice of $\dot{f}$ to represent the derivative when it is to be thought of as an operation of high precedence, but the choice of $\frac{df}{dt}$ for the derivative when its properties as a ratio are to be used.

Although it is difficult to make strict classifications, it is useful to distinguish seven distinct pictorial forms used in mathematical notation. As stated above, the more compact forms are often used for operations of higher precedence with the exception that fences and size changes permit the use of complex arguments in the simpler forms; size changes allow complex forms to have high precedence.

The forms in rough order of decreasing precedence, with some examples of each are:

1.	supersub	$U_{1'1}$; X^2 ; ${}^2T_3$
2.	concatenation with variable size fence symbols and separation symbols	fn(x, y); X ; $\begin{pmatrix} a \\ b \end{pmatrix}$; (a)
3.	concatenation	2X; XYZ
4.	binding symbol	$\frac{a+b}{c}$; $\int_a^b x dx$; $\sum_{x=3}^{6} X$
5.	hybrid	$\frac{d^3f}{d^2xdy}$
6.	concatenation with infix symbols	$x=y$; $a+b+c$; $A\cap B$; $A \cdot B \cdot C$
7.	title	(El) X

Since the same mechanisms, such as concatenation, are used in more than one of these seven forms, we will discuss the simpler ones first.

a. <u>Concatenation</u>

A rewrite rule may be associated with any operator. For example, if concatenation is used to represent multiplication then there might be a rule:

(PRD A B NIL) $\longrightarrow$ (CONCAT A B NIL)

The computation is simplified if cancatenation is considered to be a binary operator. It is therefore necessary to have recursive rules such as:

(PRD X_1 X_2 --- X_n NIL) $\longrightarrow$ (CONCAT X_1 (CONCAT X_2 --- (CONCAT X_{n-1} X_n NIL) --- NIL)NIL).

This rule, however, makes a multiple level structure out of a single level structure and thus destroys the form of the internal tree which is needed to parse the symbols for the display. To remove this difficulty an indicator is put on the property list of the CONCAT form to show that it is not to be considered a node in the internal tree when the symbols are sent to the display by pass four. A pseudo-form, DELIMIT, is introduced to correspond to the old PRD operator. The shape of DELIMIT is just the shape of its argument. The rule above then becomes:

(PRD X_1 X_2 --- X_n NIL) $\rightarrow$ (DELIMIT(CONCAT X_1 (CONCAT X_2 ---
(CONCAT X_{n-1} X_n (UNDELIMIT)) --- (UNDELIMIT)) (UNDELIMIT))NIL)

b. <u>Concatenation with variable size fence symbols and separation symbols</u>.

In the example fn(X, Y), the size of the comma does not depend on the size of the arguments X and Y so it can be handled by the form CONCAT. On the other hand, the size of the parentheses depends on the dimensions of the enclosed expression and so it is necessary to introduce a parenthesis form. The rule for functions is:

(FUNCTION X_1 X_2 --- X_n NIL) $\rightarrow$ (DELIMIT(CONCAT X_1(PAREN(CONCAT
X_2 (CONCAT(ATOM , (UNDELIMIT)) - --(CONCAT(ATOM , (UNDELIMIT))
X_n (UNDELIMIT)) (UNDELIMIT)) (UNDELIMIT)) (UNDELIMIT))
(UNDELIMIT)) NIL).

The size of the parentheses is chosen during the dimension pass. They are considered to be symbols introduced by the PAREN form, rather than arguments of the form. The dimension pass also tells the PAREN operator the depth of the parentheses contained in its argument. The PAREN operator then chooses parentheses, brackets, or braces accordingly. For example, the compiler would produce $\left\{ a \cdot \left[c \cdot (b+d) \right]^2 \right\}^3$. In the case of transendental functions, the parentheses are omitted if the argument is a single variable, a product of two variables, or a division.

c. <u>Binding symbol</u>

Often a size change is associated with a binding symbol. One might want to make the size change recursive; however, it takes only a few size changes to make the range of sizes too large. The current system uses characters which differ in size by a factor of two and in this case more than one size change is unsatisfactory. Fortunately, expressions occurring in practice would rarely require more than one size change if the recursive rule were to be used. The second pass carries the size with it as it goes out the branches of the internal tree; the rewrite rule for the mathematical operator of any node of the tree may specify that certain branches from that node are to be rewritten using the smaller size. In order to present the rewrite rules, the smaller size has been indicated by a 1 on the rewritten expressions property list. In actual fact, the size is saved on a pushdown list where it can be retrieved by the third pass. An example of a rule involving a size change is the rule for integrals where the expressions for the limits are rewritten using the smaller size:

```
(ITG X1 (ATOM A NIL) (ATOM B NIL) X2 NIL)→
(ITG (CONCAT (ATOM D (UNDELIMIT) ) X1 (UNDELIMIT) )
(ATOM A (1) ) (ATOM B (1) ) X2 NIL).
```

d. Supersub

This notation is used for exponentiation and for subscripting. The placement of subscripts is a property of the particular variable or function name. The subscripts can be placed at any of the four corners, so that there are 15 possible spatial arrangements. To avoid using 15 distinct but similar operators, null arguments have been introduced. They are treated as picture parts having zero dimensions. When the rewriting pass discovers a NAM operator is picks up a subscript-placement-list associated with the variable or function name in question. This is a list of the form (X1 X2. . . . Xn), where each Xi is one of the symbols NE, NW, SE, SW. The ith argument of NAM is then placed at corner Xi. Two or more arguments at the same corner are separated by commas. Placement of arguments must start at the NE corner and proceed counter-clockwise. When the subscript-placement-list is exhausted the remaining arguments are placed at the SE corner. For example if H has the subscript-placement-list (NW SW SW) then:

```
(NAM  X1 X2 X3 X4 (ATOM H NIL) NIL→ (SUPERSUB NIL X1
(CONCAT X2 (CONCAT (ATOM   , (UNDELIMIT) ) X3 (UNDELIMIT) )
(1 UNDELIMIT) ) X4 (ATOM H NIL) NIL).
```

If a subscripted variable name is raised to a power the exponent takes the NE position if there are no NE subscripts. The rewrite rule uses SUPERSUB and the operator PWRUP which preserves the original tree by removing the exponent from the SUPERSUB level during the dimension pass and using it for its second argument.

For example:

(PWR (NAM X1 X2 NIL) X3 NIL $\rightarrow$ (PWRUP (SUPERSUB X3 NIL NIL

X1 X2 NIL)NIL NIL).

Note that this combination of levels must be handled carefully if one desires to use a scheme where duplicate subexpressions are rewritten and dimensioned only once.

e. Concatenation with infix symbols

This form has two possible spatial representations. If it is too wide for one line, then some of the arguments can be placed on additional lines. In order to choose the correct form the arguments must be rewritten and dimensioned and the available line width must be known. The arguments and symbols are concatenated horizontally until a line is filled. Successive lines are then concatenated vertically.

In the case A + - B the connecting symbol + can be omitted. Similarly one might want to write A * B, but A(B+C), omitting the * when the parentheses remove the ambiguity. On the other hand, when the expression is continued on the next line one would still write:

$\begin{array}{l}A\\ *(B+C)\end{array}$ but $\begin{array}{l}A\\ -B\end{array}$. This problem is resolved by associating with each connecting symbol lists of those operators which cause it to be suppressed. There is a list of those operators which suppress it from the left, and a list of those operators which suppress it from the right. In addition there is a list of operators which replace the connecting symbol when a new line is started. Before an argument is rewritten and dimensioned its main operator is checked against the above lists so that the connecting symbol may be omitted if necessary.

f. Title

The title operator takes as arguments an expression and its name. The name is rewritten, dimensioned, and positioned on the left edge of the display; the expression must then fit into the remaining space.

g. Hybrid

Some operators are simply combinations of the others.

In conclusion, rewriting is controlled by the available space, the type of operator, and the size and type of its arguments. The form is dependent on the dimensions only if line width is applied as a constraint.

Dimensions

The dimension pass starts at the ends of the branches of the internal tree and works towards the base. At the ends of the branches are the individual variable names. These are either the names of special characters such as θ, or a sequence of characters to be displayed by the character generator. The names of special characters are marked with a flag and have their dimensions associated with them. The width of sequences of generated characters must be computed using their size and the fact that in the SAL language used for display $ and = are mapped into character generator case shifts and / is used as a quote character, and also the fact that some generated characters are non-spacing.

Five dimension functions must be written for each form. There are functions for the width, height, depth, symbols, and arguments. The compiler uses these functions to put the dimensions of each picture part on its property list. The three functions for the width, height, and depth have a list of the dimensioned arguments of the form as input. They use helping functions to retrieve the dimensions from the property lists of the arguments. The position of an argument or symbol is specified by the coordinates of the lower left hand corner of the rectangle containing it. The function for the arguments has as value an S-expression of the form (ΔX_1 ΔY_1 ---ΔX_n ΔY_n) where ΔX_i , ΔY_i is the position of the i^{th} argument with respect to the lower left hand corner of the circumscribed rectangle. The function for the symbols has as output an S-expression of the form (ΔX_1 ΔY_1 $NAME_1$ S_1 W_1 ----ΔX_n ΔY_n $NAME_n$ S_n W_n) where the i^{th} symbol which is added to the display by this form has name $NAME_i$, size S_i, width W_i, and relative position ΔX_i ΔY_i. The functions for the symbols and arguments have as input the width, height, and depth, as well as a list of the dimensioned arguments. The computation of the dimensions and relative positions by independent functions breaks down when symbols such as parentheses must be chosen, as it is not desirable to repeat this choice for each of the five functions. To solve this problem a mechanism is set up by which the width function, which is executed first, can communicate its choice to the others.

Each of the forms currently in the system is pictured in the following figures. Dotted lines have been drawn to indicate how the arguments and center lines are placed. It might be well to note that the human eye is sensitive to even the smallest misalignments.

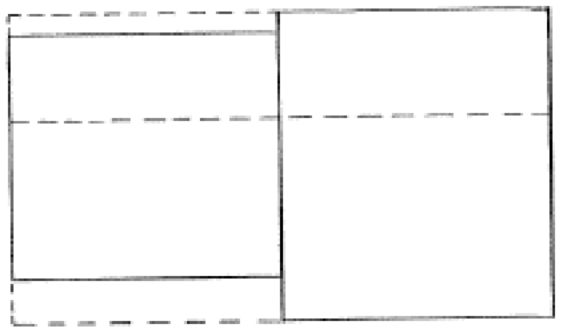


Figure 1: CONCAT

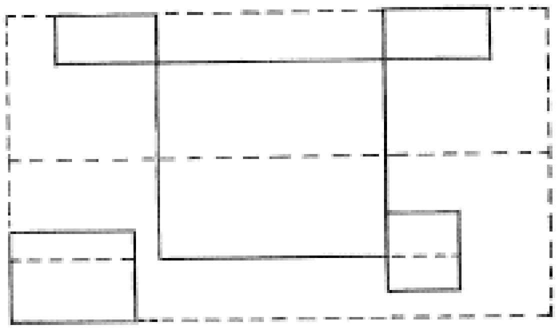


Figure 2: SUPSUB

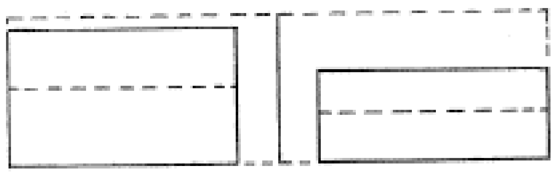


Figure 3: EVL

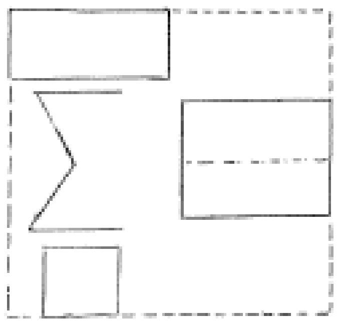


Figure 4: SUM

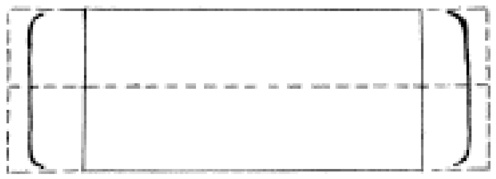


Figure 5: PAREN

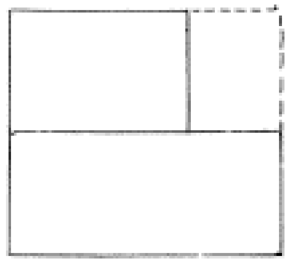


Figure 6: LVCONCAT

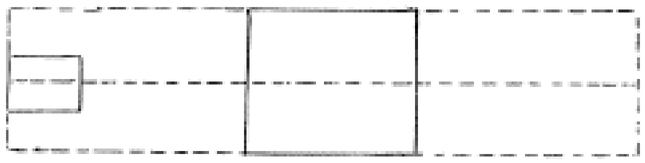


Figure 7: TITLE

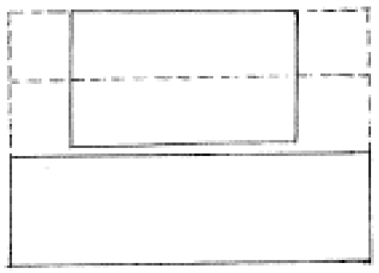


Figure 8: VCONCAT

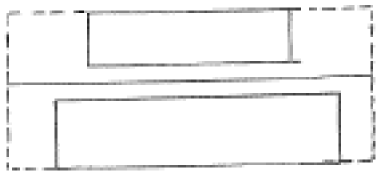


Figure 9: DVD

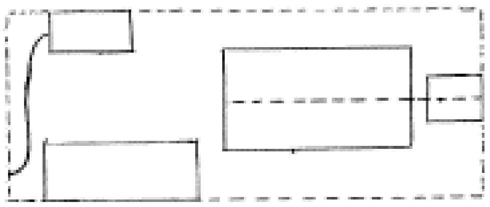


Figure 10: ITG

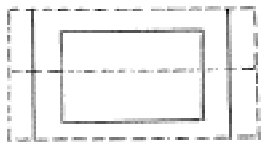


Figure 11: ABS

Output to the Display

The LISP picture compiler program runs in Project MAC time sharing. After an expression has been dimensioned, the compiler outputs a description of the picture to be displayed over the dataphone to PDP-6 LISP. The picture is displayed on the PDP-6 scope using the SAL and MACROSAL languages. Each expression picture is set up as a single MACROSAL object with the picture tree marked by left and right pseudo-parentheses. The compiler starts at the base of the tree and goes out the branches, taking them in left to right order. The compiler sends each symbol to the display as it is encountered. If two successive symbols are not adjacent, then the compiler links them with a non-displaying relative vector. Whenever the compiler encounters a new node of the internal tree it follows the last symbol sent with a left pseudo-parenthesis. After the compiler has finished this node and all branches extending from it the corresponding right pseudo-parenthesis is sent. This information is also sent to the disk so that the picture can be reconstructed without being recompiled.

<u>Example</u>

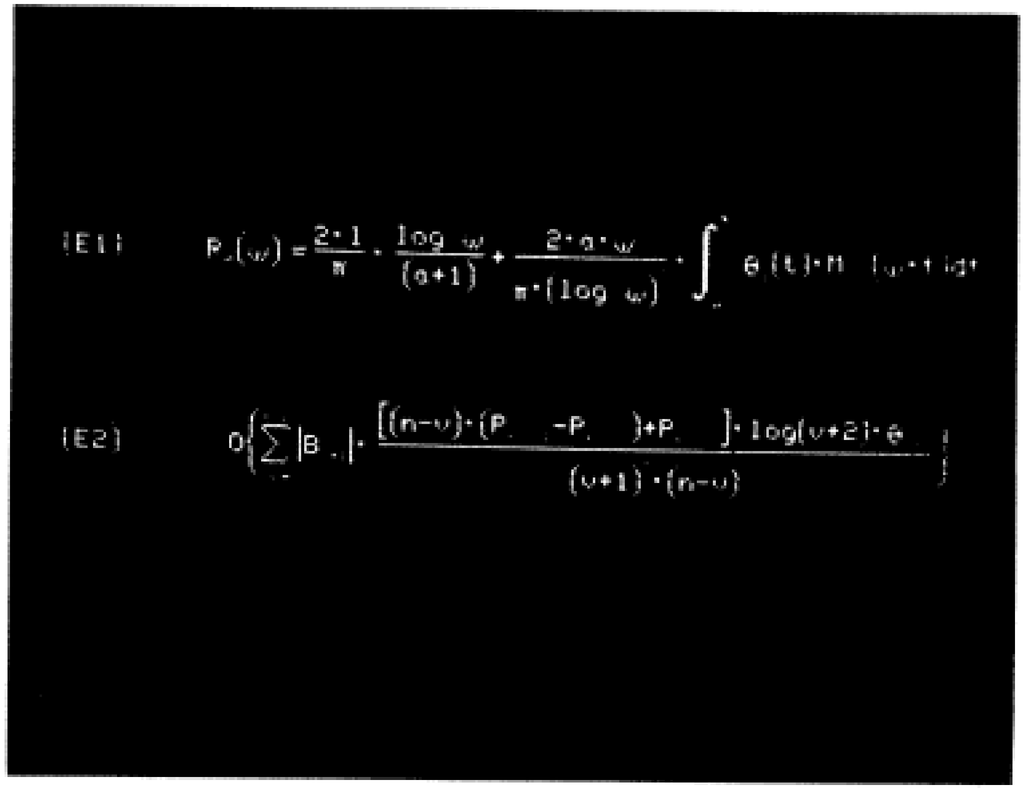
The following figure is a photograph of two displayed expressions.

The source expression for E1 is:

```
(EQN(PWR(NAM O(R OMEGA NIL)NIL)*A NIL)
(PLS(PRD(PRD 2 *L(PWR PI -1 NIL)NIL)
(PRD(*LOG OMEGA NIL) (PWR(PLS *A 1 NIL)-1 NIL)NIL)NIL)
(PRD(PRD 2 *A OMEGA (PWR PI -1 NIL) (PWR(*LOG OMEGA NIL) (PRD -1 *A
  NIL)NIL)NIL)
(ITG *T O PY (PRD(NAM 1(THETA *T NIL)NIL)
(NAM(PLS *A -1 NIL) (M(PRD OMEGA *T NIL)NIL)NIL)NIL)NIL)NIL)NIL)NIL)
```

The source expression for E2 is:

```
(O(SUM *V(PRD *N(PWR 2 -1 NIL)NIL)
(PLS *N -2 NIL) (PRD(ABS(NAM(PLS *V 1 NIL)B NIL)NIL)
(PRD(PLS(PRD(PLS *N(PRD -1 *V NIL)NIL)
(PLS(NAM(PLS *N(PRD -1 *V NIL)-1 NIL)P NIL)
(PRD -1(NAM(PLS *N(PRD -1 *V NIL)-2 NIL)P NIL)NIL)NIL)
NIL) (NAM(PLS *N(PRD -1 *V NIL) -2 NIL)P NIL)NIL)
(*LOG(PLS *V 2 NIL)NIL) (NAM(PLS *V 1 NIL)
THETA NIL) (PWR(PLS *V 1 NIL)-2 NIL)
(PWR(PLS *N(PRD -1 *V NIL)NIL)-1 NIL)NIL)NIL)NIL)NIL)
```



(E1) $$P_{\nu}(w) = \frac{2 \cdot 1}{\pi} \cdot \frac{\log w}{(a+1)} + \frac{2 \cdot a \cdot w}{\pi \cdot (\log w)} \cdot \int \theta(t) \cdot M \quad (\nu + t)dt$$

(E2) $$O\left(\sum |B_{\nu}| \cdot \frac{\left[(n-\nu) \cdot (P \quad - P \quad) + P \quad \right] \cdot \log(\nu+2) \cdot e}{(\nu+1) \cdot (n-\nu)}\right)$$

Conclusion

One measure of program complexity is the extent and predictability of the combinations of independent parts of the input data which must be made in order to determine the flow of program control. In this respect the display of mathematical expressions seems to lie midway in difficulty between the display of lines of English text and the display of arbitrary graphs. In the display of English text, the data is organized in a string, the only global property required is a character count. Hyphenation decisions are based on the word at the end of the line. On the other hand, the display of graphs seems to require the simultaneous positioning of several nodes, and might require a character algebra or an iterative approach. In the display of mathematical expressions, decisions can be made at each node of the expression tree, based on information collected from above and below. The central importance of this tree structure makes LISP a convenient language for the program.

Chapter X

LINEAR INPUT OF MATHEMATICAL EXPRESSIONS

R. W. Floyd describes an algorithm for parsing a language with a precedence grammar. Floyd defines an operator grammar as one with no productions containing adjacent non-terminal symbols. A precedence grammar is then an operator grammar in which given two ordered terminal symbols, one either appears before, at the same time, or after the other, in all series of productions leading to a string in the language. The productions shown in Figure 1 will, with three exceptions, produce the desired input strings as a subset of all the strings which they produce. With the exception of N, which stands for any digit, and L, which stands for any letter of the alphabet, the letters are the non-terminal symbols. The initial non-terminal symbol is V. The productions 10, 12, 29 and 42 all represent classes of productions formed by taking any number of S's or P's, separated by commas.

To verify that these productions form a precedence grammar, the table shown in Figure 2 is formed. For any two terminal symbols T_1 and T_2, T_1 is $>$, $<$, or $=$ T_2, if it appears before, after, or at the same time as T_2 in a series of productions leading to a string of terminal characters, and in which T_1 and T_2 are at some point in the series, either adjacent or separated only by non-terminal characters. Floyd gives a systematic way of checking a proposed set of productions for these relations. In Figure 2 it can be seen that only one of these relations holds for each ordered pair of terminal symbols, and so the productions do form a precedence grammar and a string formed by these productions can be parsed using Floyd's algorithm.

These productions produce strings which differ from the strings to be input in three ways, and so the input strings must be preprocessed. In the input strings, the unary minus is written the same as the binary one, the preprocessor recognizes it by the immediately preceding left parenthesis and changes it to theta, in order to conform with productions 20 and 43. Similarly, the quotes, ', which precede a left parenthesis are changed to double quotes, ", to agree with productions 3, 7, 42, 43 and 24. Finally the string !B, N typed at the PDP-6 is changed to B!N by the PDP-6 as it picks up the subexpression address integer N from displayed expression B.

To the parsing algorithm must be added a description of how meaning is assigned to the parsed expression. As the input string is parsed, the non-terminal symbol on the left side of each production discovered is represented by the LISP expression which is the meaning of that production. For example X + Y would be identified as an instance of production 13 and changed to (PLS X Y). The steps in parsing X + C * D would be:

```
X + C * D
X + (PRD C D)
(PLS X (PRD C D) ).
```

The function which forms the LISP expression associated with each production is found on the property list of the left most terminal symbol of the right side of the production. Sometimes, the same terminal symbol is left-most in several productions, but the production in which the symbol lies can then be determined from the other terminal symbols in the production or the total number of symbols in it. Productions 33 through 41 are tested for separately, so that a function will not have to be placed on the property list of each letter and digit. The method of assigning meaning to these productions is also different, for they must be transformed into LISP atoms and numbers. Corresponding to the meaning assignments are the rules:

$$
\begin{aligned}
L &\rightarrow \text{(ATOM L)}\\
N &\rightarrow \text{(ATOM N)}\\
\text{L(ATOM...)} &\rightarrow \text{(ATOM L....)}\\
\text{N(ATOM...)} &\rightarrow \text{(ATOM N.....)}.
\end{aligned}
$$

Then, a post-processing function is applied to the final parsed expression and the subexpressions starting with ATOM are replaced by the LISP atom or number made from the characters in the subexpression. The remaining meaning assignments are shown in Figure 3.

A flow chart of Floyd's parsing algorithm is shown in Figure 4. This chapter will be concluded with a description of the LISP program to perform this algorithm. The top level function APARSE accepts the list of input symbols, which is in reverse order, reverses it, and gives it to APARSE1 for the preprocessing step. APARSE then parses this preprocessed string, using the PROG variables U, V, P, R, and S for temporary results. U is the list of symbols to be parsed. The answer is built up on P. NIL is used for the left and right end terminal symbol. In brief, terminal symbols on P are combined into phrases, i.e. non-terminal symbols, and these phrases are combined with other terminal symbols into larger phrases until there is only one phrase, the result, between the two terminal NIL's. The first step is to put the right terminal NIL on P. Then, the next symbol, R, is taken from the input list U. When U is empty, the left terminal NIL is taken. The next step is to look down the list of phrases and terminal symbols on P, calling the terminal symbol under consideration S, until one is found with higher precedence than R. This means that S belongs to a non-terminal phrase which came down after the phrase containing R. Control is sent to B, where all terminal symbols in this phrase which have equal precedence with S, and the intervening phrases are determined. These symbols and phrases are placed on V, where they are combined into a phrase by a function associated with S. This phrase replaces its parts on P and the comparison of symbols on P with R is resumed. If S is not greater than R, then a check is made to see if R is NIL, in which case the process is complete. If R is not NIL, then R is added to P and control goes to A, where a new input symbol R is taken and the old R becomes S, since it is now at the front of P. When this process is finished, the final phrase is operated on by APARSE2 which combines the ATOM phrases into LISP atoms and numbers. Description of APARSE2 will be deferred until the functions which create the phrases have been described. As stated above, a function is associated with each terminal symbol. Numbers, POINT, and literals use the function PFN5. If the number or literal has no phrase immediately to its right, then the structure (ATOM N) is created by PFN5. Otherwise the phrase will be of this form and the new symbol is added to it.

(N   (ATOM X . . .)) becomes (ATOM N X . . . .). Most of the other operators use the function BPFN which creates a binary prefix structure from a binary infix one. (X, op, Y) becomes (op X Y). The exceptions are (X NEG Y) which becomes (PLS X (PRD -1 Y)), (X LBK Y, Z, . . . RBK) which becomes (NAM Y Z . . . X), and LP. For LP there are three cases. (LP THETA X) becomes (PRD -1 X). ((ATOM . . .) LP X, Y. . . . RP) becomes ((ATOM. . .) X Y . . .). Finally, the subscripted function case, ((NAM . . . F) LP X, Y . . . RP) becomes (NAM . . . (F X Y . . .)). The function PFN1 is used to remove the commas in case two, and the function PFN2 is used to remove the commas in case three, placing the function structure at the end as the NAM body.

In conclusion, APARSE2 will be described. APARSE2 looks for the structures (ATOM A . . .) which it changes into a LISP number if the first character A is an integer or a LISP atom if A is a literal. Literal atoms are formed by PFN6 which uses the LISP character handling functions. Numbers are formed by PFN4, the number is built up as the first member of the list of digits, the second member being combined with it until all digits have been used or until POINT is reached, indicating that the number is floating point. The digits to the right of POINT are combined into a floating point fraction by PFN41 and then added to the integer part.

1. V → C
2. V → R ← C
3. C → "(S)
4. C → S
5. S → P = P
6. S → P
7. P → "(D)
8. P → D
9. D → E
10. D → (S, S) etc
11. D → X(S)
12. D → X (S, S)etc.
13. E → E + F
14. E → E-F
15. E → F
16. F → G
17. F → F * G
18. F → F/G
19. G → H
20. G → (ΘH)
21. H → H↑I
22. H → I
23. I → X
24. I → "(D)
25. I → (D)
26. I → Y
27. I → R! Y
28. X → R[P]
29. X → R[P, P] etc.
30. X → R
31. R → 'B
32. R → B
33. Y → . Y
34. Y → N
35. Y → NY
36. B → L
37. B → LA
38. A → L
39. A → N
40. A → LA
41. A → NA
42. I → "(S, S) etc.
43. I → "(ΘH)

Figure 1
Productions for the Input Grammar

T_2

T_1	(	)	,	=	+	-	θ	*	L	.	N	/	[	]	↑	←	!	"	'
LP (	<	=	=	<	<	<	=	<	<	<	<	<	<		<			<	<
RP)		>	>	>	>	>		>				>		>	>				
CMA ,	<	=	=		<	<		<	<	<	<	<	<	=	<		<	<	<
EQN =	<	>	>		<	<		<	<	<	<	<	<		<		<	<	<
PLS +	<	>	>	>	>	>		<	<	<	<	<	<	>	<		<	<	<
NEG -	<	>	>	>	>	>		<	<	<	<	<	<	>	<		<	<	<
THETA θ	<	=							<	<	<		<		<		<	<	<
PRD *	<	>	>	>	>	>		>	<	<	<	>	<	>	<		<	<	<
L	>	>	>	>	>	>		>	<		<	>	>	>	>	>	>		
POINT .		>	>	>	>	>		>		<	<	>		>	>				
N	>	>	>	>	>	>		>	<		<	>	>	>	>	>	>		
DVD /	<	>	>	>	>	>		>	<	<	<	>	<	>	<		<	<	<
LBK [	<		=		<	<		<	<	<	<	<	<	=	<		<	<	<
RBK]	>	>	>	>	>	>		>				>		>	>				
PWR ↑	<	>	>	>	>	>		>	<	<	<	>	<	>	>		<	<	<
EASSIGN ←	<			<	<	<		<	<	<	<	<	<		<		<	<	<
PFORMREAD !		>	>	>	>	>		>		<	<	>		>	>				
DEQUOTE "	=								<										
EQUOTE '	>	>	>	>	>	>		>				>	>	>	>	>	>		

Figure 2

Precedence Table for the Grammar

2	X ← Y	→	(EASSIGN X Y)
3, 7, 24	" (X)	→	(EQUOTE X)
5	X = Y	→	(EQN X Y)
10	(X, Y)	→	(ESET X Y)
11	X(Y)	→	(X Y)
12	X(Y, Z)	→	(X Y Z)
13	X + Y	→	(PLS X Y)
14	X - Y	→	(PLS X (PRD -1 Y))
17	X * Y	→	(PRD X Y)
18	X/Y	→	(PRD X (PWR Y -1))
20	(θ X)	→	(PRD -1 X)
21	H↑I	→	(PWR H I)
25	(X)	→	X
26	X ! Y	→	(PFORMREAD X Y)
27	X[Y]	→	(NAM Y X)
28	X[Y, Z]	→	(NAM Y Z X)
31	'X	→	(EQUOTE X)
42	"(X, Y)	→	(EQUOTE (ESET X Y))
43	"(θ X)	→	(EQUOTE (PRD -1 X))

Fig. 3

Assignment of Meaning to the Productions

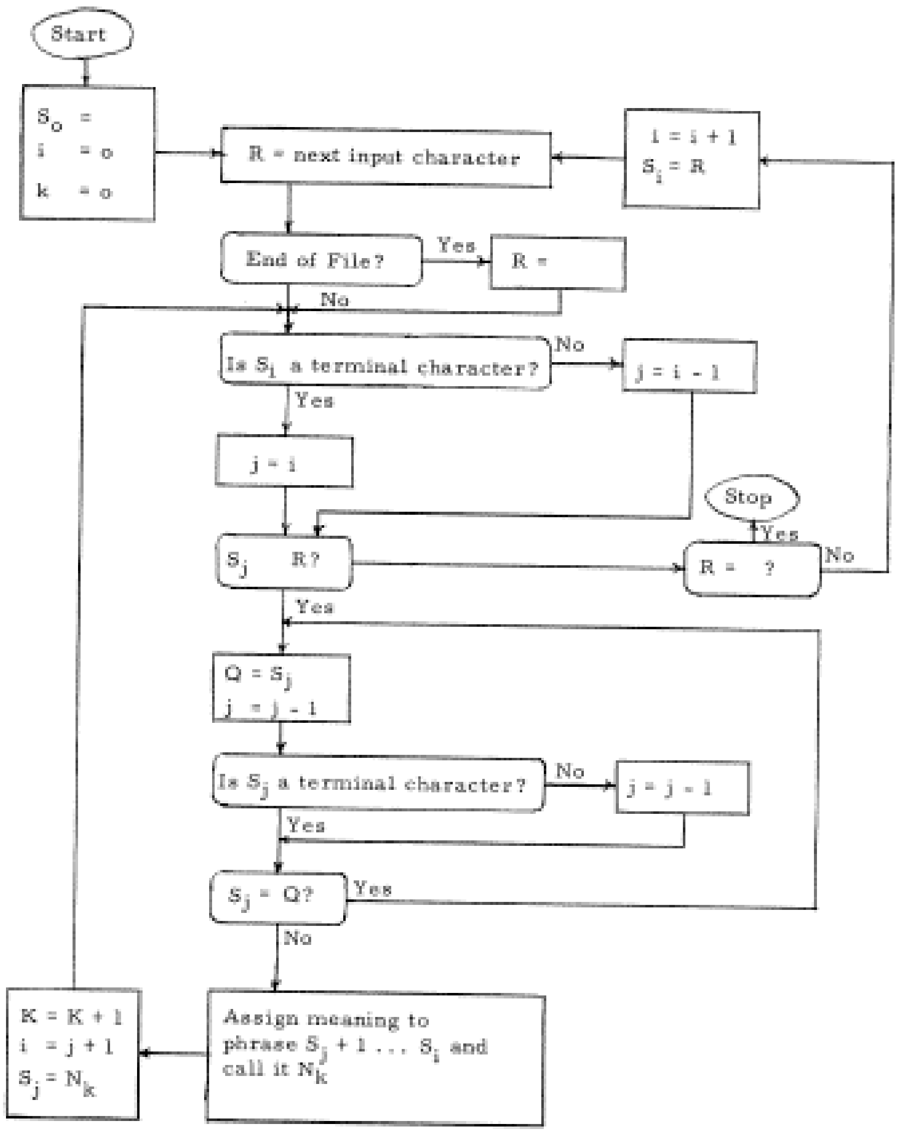


Figure 4

Floyd's Parsing Algorithm

Chapter XI

TWO DIMENSIONAL INPUT OF MATHEMATICAL EXPRESSIONS

In the preceding chapter, a mathematical expression represented in list structure form was changed into a text book style expression through the application of a series of rules. First, the order of arguments was rearranged and prefix operators became infix operators. Then, the expression was expressed in terms of two dimensional forms. Finally, the exact size and position of these forms and their corresponding symbols was determined. In this chapter, a method by which this process could be reversed will be suggested. Conditions under which the size and position of individual symbols can be used to parse an array of symbols will be given.

Klerer (4) has developed a special typewriter which allows the carriage to be spaced up and down as well as left and right. Characters typed on this machine are placed in a two dimensional array in core memory. Klerer has an algorithm for translating this array into a linear string. It would also be possible to generate characters on the display face. Teitleman (1) has demonstrated a program which recognizes individual hand printed characters. This program could be used to recognize a string of printed characters by assuming that all overlapping strokes form a single character. The program would remain several strokes behind the user, assuming that no character has more than say five strokes and that all of the strokes for a given character are made in sequence. Characters consisting of disconnected parts will be parsed together in the parsing program.

When the character is recognized, it will be replaced by a standard character in one of several standard sizes and moved into the nearest cell of an array on the display face. The user can erase and correct any false characters. The order in which the characters are generated can be saved as an heuristic aid in parsing. The characters are positioned in a matrix on the display face for two reasons. Positions such as A^B must be interpreted either as AB or A^B, the user can correct a false interpretation before parsing begins; furthermore, if characters are allowed to have any coordinates on the scope face, then the parsing program is given redundant information. The parsing program would have to consider many character positions as equivalent. This means that it does not have the possibility of terminating a string by finding no character in a given position.

The first step in parsing is to transform the two dimensional array of characters into a one dimensional string. To do this, information about the relative positions of the characters must be used. Since adjacent characters are often grouped together, it is efficient to sort the characters into a two dimensional array. Klerer uses an array with one cell for every possible typewriter position. Another possibility is to use a list structure array, with the coordinates of each character as its properties. This would take less storage if Klerer's array is nearly empty. To get the characters into a one dimensional string, it is necessary to map groups of characters with a given vertical relationship into a given horizontal relationship. For instance, the characters comprising the numerator of a division will occur before the characters comprising the denominator. Klerer identifies these groups of characters in expressions containing subscript, superscript, and binding symbol forms by imposing some constraints. First, the symbols which lie in a given area with respect to a binding symbol are taken to comprise an argument of that symbol. For example, the symbols in the numerator of a division must lie over the division bar. Second, the allowed position of input symbols is constrained so that a top to bottom, left to right scan will encounter the symbols in a cannonical order, so that every binding symbol can be found before the symbols in its arguments. Finally, superscripts and subscripts must occur in a given position with respect to the base, and occur on the right only, so that the base will be scanned first by the left to right scan.

These are not very severe constraints; but it would be possible to relax them so that the case $\frac{a + b + c}{d}$ could be handled. To do this, note that the forms can be ordered:

CONCAT

SUPSUB

LVCONCAT

VCONCAT

BINDING SYMBOL

TITLE

In addition, the binding symbols can be partially ordered. For example, if two division bars occupy the same horizontal space, the longer dominates the smaller. Once this is accomplished the one dimensional string is formed. First, all symbols related by CONCAT are found and replaced by a string with the corresponding position and dimensions. The position and dimensions are found from the definition of the CONCAT form in the last chapter. Next, all SUPSUB's are formed in the same way. If no SUPSUB's are found then all LVCONCAT's are formed. However, if some SUPSUB's are found, then CONCAT's are considered again. In this manner, each higher level form is considered. When binding symbols are considered, the form for each lowest ranking member in each group of ordered binding symbols is constructed. This process is continued until all the symbols have been converted into one linear string. Any form which can be defined for the previous chapter can be treated in this manner. Of course size must be used to distinguish A^a from $_aB$. Context dependent forms, however, cannot be handled. For example, in $\begin{pmatrix} a \\ b \end{pmatrix}$ the a and b would not be recognized to be in the VCONCAT relationship unless the vertical separation was near that specified in the last chapter. However, this relationship is implied by the large parentheses.

As stated, the above parsing scheme could be quite inefficient, since some forms might be considered several times before they were combined. It should be possible, however, to consider the expression in a systematic way, following the tree structure in most cases.

Once the expression has been converted to a linear string, it can be parsed by the algorithm given earlier for the typewriter.

Robert Anderson at Harvard is working on both a formal definition of the two dimensional syntax and a parsing scheme. He has independently suggested ideas similar to those in this chapter which he hopes to carry much farther.

Chapter XII

THE SYSTEM PROGRAMS

Three system programs were constructed as a basis for the mathematical laboratory: Time Sharing LISP, the PDP-6 LISP dataphone link, and the PDP-6 LISP display language. The development of these systems required an additional expenditure of programming effort and their use in the finished system results in some loss in both space and speed. Therefore, the description of these systems in the following chapters will be introduced by a discussion of the purpose they serve in the research and some of the factors which guided the many decisions, large and small, required for their construction.

In brief, the purpose of programming research is to investigate the structure and behavior of complex systems. Legitimate results of this research are working programs which demonstrate the realization of certain levels of technology in a given area, or program concepts which have been found useful for system description or for programming certain processes.

In practice, the development of working programs has often hinged largely on whether sufficient high calibre effort, properly organized, could be expended, rather than on the discovery of major ideas. System programs allow workers in related areas to help each other by working together to develop a system which will be used by all. Working alone results not only in the duplication of effort in writing programs to perform certain basic functions, but results in a myriad of programs which are completely incompatable because of numerous trivial differences. Voluntary collaboration may be initiated by writing a system program and offering it to the public. If this is done, then it is necessary to give the public what it wants. This is borne out by the acceptance of these systems to date. The potential users of time-sharing LISP had goals similar to the writer. The system had only one objectionable new feature, the method of writing files, while providing many new conveniences. Several people contributed additional improvements and after two years Moses and Fenichel rewrote it completely to form an excellent system which was used in this thesis. Three people have used the dataphone program; they felt the LISP system took up too much space in the PDP-6, so they expended considerable effort to provide a new environment for the program and then never did anything with it. They never got around to improving the reliability of the dataphone program, which would have benefitted the author. Occasional malfunction of the dataphone program reduced acceptance by potential casual users. Acceptance of the display language at Project MAC among the more recreational programmers was retarded when a version of PDP-6 LISP with many small improvements superceded the version in which the display language was originally imbedded. Not enough care was taken in clearly delineating the steps required to move the embedded language to a new version. Also, some of the conventions, although useful, were too controversial. Considerable interest has been shown by owners of other PDP-6 machines, who would like some display facility. Also, the embedding of I-O routines in LISP has become more popular.

System programs can also be an aid to the individual programmer. Important research programs can take a long time to construct. This leads programmers to plan far more than they can accomplish, to do work which has an incorrect emphasis in the light of the state of the art when the work is finished, and to lose time moving from one machine to another. If a programmer builds a system program as he builds his basic routines, then he is less likely to make the careless decisions which leave him inflexible with respect to the above difficulties. The important thing is not to undertake much more work because you are building a system, but to do what you do in a form suitable for a public system.

System programs are also useful in the discovery and communication of important concepts. In his recent thesis, Teitelman describes the PILOT system, which lets the programmer alter the effect of his procedures by inserting methods for handling special cases which arise as the program is used. In handling each case separately, the programmer must be careful, or he will neglect to search for a general concept which will appropriately describe all of the cases and which can be communicated to others. A programming language is a good method of communicating the many useful ideas needed to treat a new problem area.

Chapter XIII

PDP-6 LISP INPUT-OUTPUT FOR THE DATAPHONE

This chapter describes the PDP-6 LISP interface with the routine which sends and receives characters over the dataphone.

The dataphone operates in sequence break mode, requesting service each millisecond. It must be turned on and off with the LISP function dp; however, when it is running, input and output to it are handled in the same manner as to the teletype, paper tape reader, and paper tape punch. The dataphone is controlled by the control characters:

A	Write on dataphone.
C	Do not write on dataphone.
D	Read from dataphone.
E	Do not read from dataphone.

Control characters can be typed from the teletype or executed with the function cchar. Since CTSS truncates lines longer than 72 characters, a carriage return is inserted whenever 72 characters have been output to the dataphone since the last carriage return. If the user sends the ASCII character % through the dataphone program, it will be sent as the CTSS quit character, similarly ASCII & is interpreted as the CTSS interrupt character. There are several LISP functions written in machine language which are useful for dataphone programming:

dp (x)	If x is NIL the dataphone is turned off; otherwise it is turned on and initialized.

dpd (x)	The value of dpd (x) is a flag for characters in the dataphone input buffer. If x is NIL, dpd returns NIL if there are no characters; otherwise, it waits until some arrive. When characters are received this function returns a list of the mode of the characters and the number of characters. The mode is a number determined by the sender. Characters sent from CTSS in the same manner as to a normal console are assigned mode 1.
ttyd ( )	Returns the number of characters in the teletype input buffer or NIL if there are none.
cchar (x)	Executes the lower case ASCII character x as a control character.

To establish communication with CTSS, turn on the dataphone by executing dp (T). Then execute ctss ( ) and type S followed by a space. The teletype can now be used as a CTSS console. Several useful features of the LISP functions ctss, dpwrite, and dpread are described below. LISP expressions for these functions are at the end of the chapter.

dpread ( )	This function waits until there is input from the dataphone. It then prints the characters on the teletype one by one. If it receives $, it does not print this, but instead evaluates the S-expression which follows.
dpwrite (x;y)	*dpwrite* takes two forms of input. If y is NIL *dpwrite* assumes that x is a list of atoms and sends over the dataphone the characters in the PNAMES of these atoms, with a space between each PNAME. Otherwise it sends x as an S-expression.
ctss ( )	*ctss* allows the user to operate the PDP-6 as a CTSS console. There are two modes; local and send. *ctss* is initially in local mode. In this mode it accepts S-expressions for *eval* but watches for the single atom S-expressions S, STOP, and L. If it finds S it goes into send mode. If it finds STOP it terminates returning NIL. If it finds L it sends over the dataphone the two S-expressions which are typed next,

but stays in local mode.

In send mode it sends characters one by one while watching for @ and #. @ returns it to local mode. If it sees # it sends the evaluation of the S-expression which follows and then types #.

```
(DEFLIST ( (CTSS (LAMBDA NIL (PROG (U V) A (COND ( (TTYD NIL) (GO
B) )  ( (DPD NIL) (DPREAD) ) ) (GO A) B (SETQ U (READ) ) (TERPRI) (COND
( (EQUAL U (QUOTE S) ) (GO C) ) ( ( EQUAL U (QUOTE STOP) ) (RETURN NIL) )
( (EQUAL U (QUOTE L) ) (PROG2 (DPWRITE (READ) T) (DPWRITE (READ)
T) ) ) (T (PRINT (EVAL U NIL) ) ) ) E (TERPRI) (GO A) C (COND ( ( DPD NIL)
(DPREAD) ) ( (TTYD NIL) (GO D) ) ) (GO C) D (SETQ U (READCH) ) (COND
( (EQUAL U@ ) (GO E) ) ( (EQUAL U #) (GO G) ) ) (CCHAR (QUOTE A) )
(CCHAR (QUOTE W) ) (PRIN1 U) (CCHAR (QUOTE C) ) (CCHAR (QUOTE V) )
(GO C) G (DPWRITE (EVAL (READ) (CDR ALIST) ) T) (PRIN1  #) (GO C) ) ) ) )
EXPR)

(DEFLIST ( (DPWRITE (LAMBDA (X Y) (PROG (U) (CCHAR (QUOTE A) ) (CCHAR
(QUOTE W) ) (COND (Y (GO C) ) ) (SETQ U X) A (COND ( (NULL U) ( GO B ) ) )
(PRIN1 (CAR U) ) (PRIN1 BLANK) (SETQ U (CDR U) ) (GO A) C (PRINT X)
B (TERPRI) (CCHAR (QUOTE C) ) (CCHAR (QUOTE V) ) (RETURN NIL) ) ) ) )
EXPR)

(DEFLIST ( (DPREAD (LAMBDA NIL (PROG (U V) (SETQ U (DPD T) ) (CCHAR
(QUOTE D) ) A (SETQ U (CADR U) ) B (COND ( (ZEROP U) (GO C ) ) ) (SETQ V
(READCH) ) (COND ( (EQUAL V S) (GO D) ) ) (PRIN1 V) (SETQ U (PLUS U
68719476735) ) (GO B) D (TERPRI) (EVAL (READ) NIL) C (CCHAR (QUOTE E) )
(RETURN NIL) ) ) ) ) EXPR)
```

Chapter XIV

PDP-6 LISP INPUT-OUTPUT FOR THE DISPLAY

A System for Display Language Construction

The system provides a language macrosal for the generation of picture parts called objects. An object can be any combination of points, lines and characters. An object is generated by calling the function macrosal [NAME; DESCRIPTION] described in the second section. The most common way to describe an object is to establish a set point. The set point is established utilizing PARAMETER, LOCY, and LOCX statements. The exact format of these statements is discussed below. An object description is terminated with a STOP statement. If NAME is T, the current description will be appended to the description of the last object generated. The first example, disp, generates a large object in this manner.

The user communicates with a display through a light pen. As the light pen sweeps across the screen, its trajectory can be used in many different ways. For example, it may be used to determine a point, a set of points, or a line. Or if a subpart of the display has been defined as an object, the trajectory may be simply interpreted as a pointer to this object or a point on the object. The LISP functions embedded in the display language facilitate acquiring the data needed to make these different levels of interpretation.

One problem in utilizing the light pen is to determine when it is near the screen and not just being moved into place. This is solved by using the width of the field of view of the pen as measured by a tracking cross. This width decreases as the pen approaches the screen and a center dot is displayed whenever the field of view is less than a certain prescribable limit. This pen distance is available to LISP.

The approach of the pen to an object is considered significant. Just how close the pen must come before being noticed is a program variable. Its most recent position within this distance is recorded; in addition, so are the last 5 such positions, each at least a prescribed distance from the preceding one. This represents a crude way of gradually forgetting the details of the past. It is also possible to get the current coordinates of the pen, obtain a list of all objects currently seen by the pen, to report when the pen sees an object with a name other than a given name, or to require an object to move with the pen.

The example function sketch is a LISP function using several of these features. The function uses a subroutine to display five different light buttons. If the light pen is held near one of these buttons, the tracking cross will be centered about the point where the pen is seen. The program interprets pointing at these buttons to mean 1) draw a line, 2) move a line, 3) delete a line, 4) suppress the cross and 5) return control to the teletype. To draw lines touch the first button, a new line will then be drawn whenever the pen leaves the screen and then returns. This process is terminated whenever the pen returns near one of the light buttons. Additional LISP functions could be written to expand sketch into a program similar to SKETCH PAD.

Often one wants to communicate to the display certain basic forms which are to be used in constructing larger units. These inputs might be a set of letters which are to be parsed into a sentence; or they might be a set of circuit elements or music symbols. In SKETCH PAD this is done by indicating the type of form and then moving the light pen so that the parameters of the form, the end points of a line segment for instance, can be abstracted from the trajectory. Alternatively, one could abstract both the type and the parameters from the trajectory. The example program <u>argus</u> uses a method developed by Teitelman which enables the user to teach the machine to replace a single line by a known form. A line is a single movement (however complex) on the surface of the display without lifting the pen. The LISP data structure is convenient for storing properties of the forms to be recognized.

The parsing of very large displayed expressions, such as LISP S-expressions for example, can be difficult for people. Furthermore, there may be alternative parsings. People can be aided by intensifying, upon request, grammatical subexpressions or sub-objects containing referenced segments or by providing additional displays meaningfully related to the first display. These might be rotated views of an object or shaded objects. Further development of the system is needed in this area.

It is the task of the programmer to organize a program and data base in such a way that the most needed inputs to a machine will have short representations and the most needed computations will be efficient. The combinatorial aspects are such that this must be done through a series of levels of concepts. An important point is that it is not possible to complete an entire level at a time. The most useful concepts at a given level only become clear with the exploration of higher levels. The exploration of higher levels without intermediate concepts is, however, almost impossibly tedious. It is important in an experimental situation to have a system where one can make changes to any desired depth and provide for the irregular growth and reorganization of the data base.

In the present display system, experimentation will probably indicate that new statements for macrosal are needed, or that certain objects occur so often that more programming effort could well be spent in generating them efficiently. To provide for these possibilities macrosal has been programmed as a syntactic extension of the scope assembly language, sal. Sal is a LISP function which creates objects from lists of octal numbers. It is described in detail in part II. Provision has also been made for the addition of machine language subroutines which alter the objects as they are displayed. Furthermore, the system is organized so that no statements need be made about features of the display language which are not needed.

By embedding these display facilities in LISP one makes available a wealth of mechanisms which have proved useful in the analysis and generation of language and in the development of systems which can be increased incrementally in complexity.

Implementation

The use of LISP in this system has two distinct disadvantages. First, it is not possible to interrupt the LISP system at any point in time and immediately employ its full power. It may be in the midst of garbage collection. Garbage collection with the current version of PDP-6 LISP requires a noticable time. Second, the data types are too limited. It is not convenient to set up the type of list structure used in SKETCH PAD, but this can be approximated. A serious problem is the inability to set aside blocks of registers to contain display instructions and to store information about light pen actions.

To get around these problems a fixed buffer of 2048 words has been set aside for description of the display. All communication between LISP and the display goes through this buffer. This buffer contains two kinds of data structures; display lists and headers. One header is associated with each display list, which is a list of half word commands for the display. The headers build down from the top of the buffer and the display lists build up from the bottom.

During display an interrupt routine cycles through a dispatch table. Dispatches can occur to a pen track routine, a routine which displays the contents of the display buffer, a line drawing routine, and a routine which terminates the display.

The pen track routine displays a cross as was described earlier. In a crude effort to give the routine enough display time each cycle, it is called between display of each object in the display buffer. Use of the clock would be better.

The format of the headers in the display buffer is as follows:

A = ON-OFF bit

B = Tracking Cross jump bit

C = Move with pen bit

NAME (Pointer to an atom)	
- (display list length	pointer to start of display list - 1
Pen hit distance	Pen hit count
Subr address	A B C
Most recent Y	Most recent X
Y	X
Y	X
Y	X
Y	X
Y	X

Figure 1 Header Format

Each display list has a name which is kept as the first word of its header. When an object is referenced by a LISP function, the headers are searched for one with the name mentioned.

The interrupt program cycles through the headers. It picks up a pointer for a BLKO instruction from the right half of the second word of each header. This BLKO is terminated by a STOP instruction at the end of each display list. If the subr address is not 0, then the subr at this address will be executed when the STOP is reached.

If the light pen is seen during display of some display list, control goes immediately to an interrupt program. Several conditional branchings can occur within this program. The interrupt program first re-displays the display list as a check against light pen noise. If the pen is seen a second time the pen cross movable bit is checked. If this bit is a 1 the pen tracking cross is centered about the point where the light pen was seen and the rest of the display list is displayed. Otherwise, a check is made to see if the center of the pen cross is within a specified minimum distance of the point seen. If not, the display list is continued with the light pen reenabled. If the pen is close enough, the coordinates of the point seen are stored in the fifth word of the header. The last five words of the header contain the coordinates of points seen by the pen in the past. Each of these history points is at least a specified distance from the preceding one. This distance is in the left half of the third word of the header. When a new point is seen, it is added to the history points if it is far enough away from the one most recently stored or if there are none. The number of history points is kept in the right half of the third word of the header. This number can be set to zero by LISP. Whenever there are more than five history points the oldest one is lost. After the point seen has been appropriately stored, a check is made to see if the object should move with the pen. If bit C is set, a pointer to the display list is transmitted to the pen track routine. The display list is then finished.

Display lists are put into the buffer by the sal language. If a line is to be drawn with the pen. A set point for the line is created with the sal language. Its display list is then incremented by the line drawing routine. This incrementing is terminated when the pen leaves the screen.

Display Functions

macrosal [X;Y]	macrosal interprets its arguments and then calls sal on the result. X is the first argument for sal. Y is a description of a display list for sal. Y is a list of lists, each of which is a macro. The first word of each macro is an atom which has under the property MACROSAL a function of one argument. macrosal gets this function and applies it to the remainder of the macro list. The result of this function is a list for sal beginning and ending in mode 1. This list is prefaced by two integers which give respectively the mode that the display will end up in and the mode in which it must begin, if it is to interpret this list correctly. macrosal appends the successive macro expansions, using the prefacing integers to create the proper linking of modes.

The following macro word formats are in the system:

1. (PARAMETER penenable scale intensity)

   Normally the first word in a set point, this statement creates a parameter word. If penenable, scale, or intensity is NIL, the corresponding field of the parameter word is not enabled.

2. (LOCY n)

   Creates a non-displaying Y point word which sets the scope Y coordinate to n. This is normally the second statement in a set point.

3. (LOCX n)

   Creates a non-displaying X point word which sets the scope X coordinate to N. This is normally the third and last statement in a set point.

4. (STOP)

   Creates a parameter word with stop enabled. This is normally the last statement in a display list.

5. (ISTOP)

Creates a parameter word with stop enabled and a flag that the previous word has a breakout bit. The use of ISTOP is explained in the description of <u>sal</u> which follows.

6. (LOCYD n)

Creates a displaying Y point word which sets the display Y coordinates to n and displays a point.

7. (LOCXD n)

Creates a displaying X point word which sets the display X coordinate to n and displays a point.

8. (LINE X_1 Y_1 --- X_n Y_n)

Creates a sequence of non-displaying line segments from vector words. The display starts at the last point displayed or set by LOCX, LOCY. Each increment has size $X = X_{i+1} - X_i$; $Y = Y_{i+1} - Y_i$.

9. (LINED X_1 Y_1 --- X_n Y_n)

Creates a line like LINE, but displays it.

10. (LONGLINE X_1 Y_1 X_2 Y_2)

Creates a non-displaying vector continue word. A line will be drawn from the current display coordinates to the edge of the display. The slope of the line is $(Y_2 - Y_1)/(X_2 - X_1)$.

11. (LONGLINED X_1 Y_1 X_2 Y_2)

Creates a line like LONGLINE, but displays it.

12. (CHAR X_1 ----X_n)

Creates a display of character made words which are the PNAME's of the atoms X_i. No spaces are inserted between the PNAME's.

sal [X;Y;Z] X is the name of the display list to be created or T. If X is T, then this list is appended to the last one created. If the previous list ends in STOP (30000_8), the STOP is removed. If it ends in ISTOP (4030000_8), it not only removes this but zeros the breakout bit (400000_8) in the previous half word. Y is a description of the list to be created. There are two forms for the elements of Y corresponding to two

modes for the assembly function sal.
The function is initially in mode 1.
In mode 1 sal removes lists of atoms, two
at a time, from Y. Each atom is a number
or is bound to one on the dotted pair list
Z. For each pair of lists sal forms one
display half word instruction by shifting
the numbers on the second list the number
of places specified by the corresponding
number on the first list. When it encounters
NIL on Y, it goes into mode 2. In mode
2 sal takes numbers or non-numerical
atoms one at a time from Y. Each number
is a display half word. If a non-numerical
atom is encountered, sal looks at the
previous number to see if it puts the display
into increment or character mode. If in
character mode, sal assembles the PNAME
of the atom as characters. If in increment
mode, sal gets a list of full words off the
atom's property list with the indicator SCHAR.
It assembles these as increment mode half
words. Consecutive non-numerical atoms
are assembled together in the same mode.

When sal finds NIL, it returns to mode 1. Mode 1 is more flexible and mode 2 is more economical. NIL is not a legal object name.

More Functions

In the description below S stands for the name of an object with a set point. W stands for the name of an object with or without a set point.

sx [S]	Returns the current x coordinate of S.
sy [S]	Returns the current y coordinate of S.
smv [S]	Makes object S follow the light pen whenever S sees it. Returns S.
sunmv [S]	Negates smv. Returns S.
sxyinc [S;X;Y]	Increments the set point coordinates of S by X, Y. Returns S.
sclr [ ]	Clears the display buffer. Returns NIL.
sdlt [W]	Deletes object W from the display buffer and returns W.
ptrk [X;Y]	Starts the pen tracking cross at X, Y and returns NIL.
puntrk [ ]	Stops the display of the pen track cross. Returns NIL.
px [ ]	Returns the x coordinate of the pen.
py [ ]	Returns the y coordinate of the pen.

ptchp [ ]	Returns NIL if the pen is far from the screen.
srn [ ]	Starts the display build up in the buffer. Display will continue until sunrn [ ] is called.
sunrn [ ]	Stops the display and returns NIL.
pldw [X;Y]	To use this command, use macrosal to set up a set point at the current pen position followed by a zero length relative line and terminated with ISTOP. Then call pldw. Increments of length at least X and scale Y will be added to this object until the pen leaves the screen. Returns NIL.
phclr [W]	Clears the pen approach history points of object W and returns W.
ph [W]	Returns a list of the pen approach history points for object W.
phc1 [W;X]	Sets to X the minimum distance between pen approach history points for object W. X is an integer between 0 and 1024.
phc2 [X]	Sets to X the minimum distance between pen approach history points which is assumed for newly created objects.

phc3 [X]	Sets to X the maximum distance on the screen at which the pen can see an object. X is an integer between 0 and 40.
sint [W;Y]	Sets to Y the intensity of object W. Returns W. W is an integer between 0 and 7.
scopy [W1, W2]	Copies object W1 and names the copy W2. Returns W1
plsh [ ]	Returns a list of all objects currently being seen by the pen.
sscl [W;Y]	Sets the scale of object W to Y. Returns W. Y is an integer between 0 and 3.
plhw [W]	Returns the name of the first object other than W seen by the pen.
ptchw [ ]	Waits until the pen is near the screen and then returns NIL.
puntchw [ ]	Waits until the pen is not near the screen and then returns NIL.
pjp [W]	If the tracking cross is not following the pen and pjp [W] has been executed, then when the pen sees object W, the tracking cross will be centered about the point of W seen. Returns W.
punjp [W]	Negates pjp and returns NIL.

pch [ ]	Returns the name of the most recent object to be seen by the pen or NIL if none have been seen since pohc was executed.
pohc [ ]	Sets the last object seen by the pen to NIL. Returns NIL.
sline [S]	Returns a list of the coordinates of the end points of the line segments which make up object S.
ssubr [W;Y]	Causes the subroutine beginning at location Y to be executed each time object W is displayed.
sloc [W]	Intensifies a subexpression of object W which is indicated by the light pen. Subexpressions are marked in the object by pseudo parentheses which are not displayed. A pseudo left paren is indicated by the parameter word 600001_8 and a pseudo right paren by the parameter word 600002_8.
sadd [W]	Counts the number of pseudo left parens to the intensified subpart of W.

```
(PRINDEF DISP MV EXPOND)
(DEFLIST ( ( DISP (LAMBDA (N) (PROG (U V)(MACROSAL (QUOTE DISP)
(QUOTE ( (PARAMETER 1 2 3) (LOCY 1) (LOCX 1) (LINED 0 0 0 0)
(ISTOP) ) ) ) (SETQ U (EXPAND N NIL) ) A (COND ( (NULL U) (RETURN NIL) ) )
(MACROSAL T (LIST (COND ( (EQUAL (CAR U) (QUOTE U) ) (QUOTE (LINED
0  0  0 4 ) ) ) ( (EQUAL (CAR U) (QUOTE D) ) (QUOTE (LINED 0 0 0 777777777774) ) )
( (EQUAL (CAR U) (QUOTE L) ) (QUOTE (LINED 0 0 777777777774 0) ) )
(T (QUOTE (LINED 0 0 4 0) ) ) ) (QUOTE (ISTOP) ) ) ) (SETQ U  (CDR U) )
(GO A) ) ) ) ) EXPR)

(DEFLIST ( (MV (LAMBDA (X) (COND ( (EQ X (QUOTE U) ) (QUOTE D) )
( (EQ X (QUOTE D) ) (QUOTE U) ) ( (EQ X (QUOTE L) ) (QUOTE R) )
(T (QUOTE L) ) ) ) ) ) EXPR)


(PRINDEF EXPAND)
(DEFLIST ( (EXPAND (LAMBDA (N EXP) (PROG (A B C D E ) (SETQ EXP
(QUOTE (U R D R) ) ) START (COND ( ( ZEROP N) (RETURN EXP ) ) ) (SETQ N
(PLUS N -1. 0) ) LOOP (SETQ A (CAR EXP) ) (SETQ EXP (CDR EXP) ) SETOB
(CAR EXP) ) (SETQ EXP (CDR EXP) ) (SETQ C (CAR EXP) ) (SETQ E
(NCONC E (LIST B A (MV B) A A B C B (MV C) B C C (MV B) C B D) ) )
(COND (EXP (GO LOOP) ) ) (SETQ EXP E) (SETQ E NIL) (GO START) ) ) ) )
EXPR)
```

```
(PRINDEF SKETCH)

(DEFLIST ( (SKETCH (LAMBDA (N X) (PROG (U W) (COND (X (GO A) ) ) (PJP
(BUTTON (QUOTE S)  1ØØ 17ØØ) ) (PJP (BUTTON (QUOTE D) 3ØØ 17ØØ) )
(PJP (BUTTON (QUOTE M) 5ØØ 17ØØ) ) (BUTTON (QUOTE P) 7ØØ 17ØØ)
(PJP (BUTTON (QUOTE R) 11ØØ 17ØØ) ) A (POHC) (SETQ W (PLHW NIL) )
B (COND ( (MEMBER W (QUOTE (D M) ) ) (GO C) ) ( (EQUAL W (QUOTE S) )
(GO G) ) ( (EQUAL W (QUOTE R) ) (GO H) ) ( (EQUAL W (QUOTE P) ) (GO I) )
( (EQUAL U (QUOTE M) ) (GO E) ) ( (EQUAL U (QUOTE D) ) (SDLT W) ) )
(GO A) C (SDLT U) (SETQ U W) (BUTTONON W) (SETQ W (PLHW W) ) (GO B)
E (SMV W) (PUNTCHW) (SUNMV W) (GO A) G (SDLT U) (BUTTONON W)
(SETQ U W) D (PUNTCHW) (PTCHW) (COND ( (GREATERP (PY ) 16ØØ) (GO A) ) )
(LINEDRAW (GENSYM) N 1) (GO D) H (SDLT U) (RETURN NIL) I (SDLT U)
(SETQ U W) (BUTTONON W) (PUNTRK) (GO A) ) ) ) ) EXPR)


(PRINDEF LINEDRAW BUTTON BUTTONON BUTTONON1)
(DEFLIST ( (LINEDRAW (LAMBDA (NAME QUALITY SCALE) (PROG NIL(PTCHW)
(SAL NAME (APPEND (QUOTE ( (Ø) (Ø) (Ø 4) ) ) (CONS (LIST 34114 SCALE)
(CONS (LIST Ø) (APPEND (CONS (LIST (PLUS 22ØØØØ (PY) ) ) (CONS
(QUOTE (Ø)) (LIST (LIST (PLUS 114ØØØ (PX) ) ) ) ) ) (QUOTE ( (Ø) (4Ø Ø Ø Ø Ø)
(Ø) (5ØØØ) ) ) ) ) ) )  NIL) (PLDW QUALITY SCALE) (RETURN NAME) ) ) ) ) EXPR)

(DEFLIST ( (BUTTON (LAMBDA (NAME X Y) (SXYING (SAL NAME (QUOTE ( (
ø)  (ø) (ø) (34114) (ø) (22øøøø)  (ø) (114øøø ) (ø) (2øøø1ø) (ø) (3

ø4øøø) (ø) (2øø21ø) (ø) (6ø4øøø) (ø) (3øøø) ) ) NIL) X Y) ) ) ) EXPR)


(DEFLIST ( ( BUTTONON (LAMBDA (NAME) (BUTTONON1 NAME (SX NAME) (SY

 NAME ) ) ) ) ) EXPR)


(DEFLIST ( (BUTTONON1 (LAMBDA (NAME X Y)  (SXYINC (SAL NAME (QUOTE

( (ø) (ø) (ø) (34114) (ø) (22øøøø) (ø) (114øøø) (ø) (7ø4ø18) (ø)

(3øøø) ) ) NIL) X Y) ) ) ) EXPR)
```

```
(PRINDEF ARGUS MAKSTRING MAKSTRING1 MAKSTRING2
MAKSTRING3)
DEFLIST ( (ARGUS (LAMBDA NIL (PROG (U V R Q MERGELIST NQ) (SETQ U
(LINEDRAW (QUOTE ARGUS) 2 1) ) (PUNTCHW) (SETQ V (MAKSTRING
(SLINE U) ) ) (SETQ R (MAP2 V ATREES (QUOTE GETC) ) ) (MAP2 AWEIGHTS R
(FUNCTION (LAMBDA (WEIGHT LIST) (MAP LIST (FUNCTION (LAMBDA
(CANDIDATE) (SETQ MERGELIST (MERGE3 MERGELIST) ) ) ) ) )) ) (COND
( (NULL MERGELIST) (GO A) ) ) (SETQ NQ (CDAR MERGELIST) ) (SETQ Q
(CAAR MERGELIST) ) (MAP MERGELIST (FUNCTION MERGE1) ) A (SDLT U)
(MACROSAL (QUOTE LAST) (LIST (QUOTE (PARAMETER 1 3 3 ) ) (LIST
(QUOTE LOCY) AYMIN ) (LIST (QUOTE LOCX) AXMIN) (LIST (COND ( (GET Q
(QUOTE SCHAR) ) (QUOTE SCHAR) ) (T(QUOTE CHAR) ) ) (COND ( (NULL Q)
(QUOTE ?) ) (T Q) ) ) (QUOTE (STOP) ) ) ) (COND ( (NOT (EQUAL AMODE
(QUOTE TRAIN) ) ) (RETURN Q) ) ) (PRINT R) (SETQ U (READ) ) (COND
( (NOT (EQUAL Q U) ) (GO B) ) ) (CSETQ AWEIGHTS (MAP2 AWEIGHTS R
(FUNCTION (LAMBDA (X Y) (COND ( (MEMBER U Y) (PLUS X 1) ) (T X ) ) ) ) ) )
B (CSETQ ATREES (MAP2 V ATREES (QUOTE PUTC) ) ) (RETURN U) ) ) ) ) EXPR)

(DEFLIST ( (MAKSTRING (LAMBDA (LINE) (PROG (U V MIN MAX V1 V2)
(MAKSTRING2 (EVERYOTHER LINE) ) (CSETQ AXMIN MIN(  MAKSTRING2
(EVERYOTHER (CDR LINE) ) ) (CSETQ AYMIN MIN) (RETURN U) ) ) ) ) EXPR)

(DEFLIST ( (MAKSTRING1 (LAMBDA (X) (COND ( (GREATER P (CAR X) V2)
(LIST NIL (MAKSTRING3 X NIL) ) ) (T (LIST (MAKSTRING3 X NIL) NIL) ) ) ) ) )
EXPR)

(DEFLIST ( (MAKSTRING2 (LAMBDA (X) (PROG NIL (SETQ MIN( CAR X) )
(SETQ MAX (CAR X) ) (MAP X (QUOTE MINMAX) ) (SETQ V (QUOTIENT (PLUS
MAX (MINUS MIN) ) 3) ) (SETQ V1 (PLUS MIN V) ) (SETQ V2 (PLUS V1
(QUOTIENT V 6) ) ) (SETQ U (APPEND (MAKSTRING1 X) U) ) (SETQ V1 (PLUS
V1 V) ) (SETQ V2 (PLUS V2 V) ) (SETQ U (APPEND (MAKSTRING1 X) U) )
(RETURN NIL) ) ) ) ) EXPR)

(DEFLIST ( (MAKSTRING3 (LAMBDA (X LEFT) (COND ( (NULL X) 1) ( (OR
(AND LEFT (GREATERP (CAR X) V2) ) (AND (NOT LEFT) (GREATERP V1
(CAR X) ) ) ) (PLUS 1 (MAKSTRING3 (CDR X) (NULL LEFT) ) ) ) (T MAKSTRING3
(CDR X) LEFT) ) ) ) ) ) EXPR)
```

```
 (PRINDEF MAP MAP2 SUB1 EVERYOTHER MINMAX MERGE1
(MERGE3 GETC PUTC)
(DEFLIST ( (MAP (LAMBDA (X FN) (COND ( (NUL X) NIL) (T (CONS (FN
(CAR X) ) (MAP (CDR X) FN) ) ) ) ) ) ) EXPR)

(DEFLIST ( (MAP2 (LAMBDA (X Y GN) (COND ( (NULL X) NIL) (T (CONS
(FN (CAR X) (CAR Y) ) (MAP2 (CDR X) (CDR Y) FN) ) ) ) ) ) ) EXPR)

(DEFLIST ( (SUB1 (LAMBDA (X) (PLUS 68719476735 X) ) ) ) EXPR)

(DEFLIST ( (EVERYOTHER (LAMBDA (X) (COND ( (OR (NULL (CDR X) )
(NULL (CDDR X) ) ) (LIST (CAR X) ) ) (T (CONS (CAR X) (EVERYOTHER
(CDDR X) ) ) ) ) ) ) ) EXPR)

(DEFLIST ( (MINMAX (LAMBDA (X) (COND ( (GREATERP X MAX) (SETQ MAX X) )
( (GREATERP MIN X) (SETQ MIN X) ) (T NIL) ) ) ) ) EXPR)

(DEFLIST  ( (MERGE1 (LAMBDA (LIST) (COND ( (GREATERP (CDR LIST) NO)
(PROG2 (SETQ NO (CDR LIST) ) (SETQ ) (CAR LIST) ) ) ) (T NIL) ) ) ) ) EXPR)

(DEFLIST ( (MERGE3 (LAMBDA (MERGELIST) (COND ( (NULL MERGELIST)
(LIST (CONS CANDIDATE WEIGHT) ) ) ( (EQUAL CANDIDATE (CAAR MERGELIST) )
(CONS (CONS CANDIDATE (PLUS WEIGHT (CDAR MERGELIST) ) ) (CDR
MERGELIST) ) ) (T (CONS (CAR MERGELIST) (MERGE3 (CDR MERGELIST) ) ) ) ) ) )
) EXPR)

(DEFLIST ( (GETC (LAMBDA (PSTRING TREE) (COND ( (NULL PSTRING) TREE)
( (NULL TREE) (COND ( (ZEROP PSTRING) (LIST (LIST U) ) ) (T (CONS NIL
(PUTC (SUB1 PSTRING) NIL) ) ) ) ) ( (ZEROP PSTRING) (COND ( (MEMBER U
(CAR TREE) ) TREE) (T (RPLACA TREE (CONS U (CAR TREE) ) ) ) ) ) )
(T (CONS (CAR TREE) (PUTC (SUB1 PSTRING) (CDR TREE) ) ) ) ) ) )) ) EXPR)
```

APPENDIX A.

LISTING OF THE LISP PROGRAMS

```
DEFINE
(((APARSE (LAMBDA (X) (PROG (U V P R S) (SETQ U (APARSE1 (REVERSE X))) (
SETQ P (LIST NIL)) A (COND ((NULL U) (SETQ R NIL)) (T (SETQ R (CAR U))))
 (SETQ U (CDR U)) D (COND ((ATOM (CAR P)) (GO C))) (SETQ V (LIST (CAR P)
)) (SETQ P (CDR P)) C (SETQ S (CAR P)) (COND ((PGRT S R) (GO B)) ((NULL
R)(RETURN(APARSE3(APARSE2(CAR V))))))(SETQ P(CONS R(APPEND V P)))
(SETQ V NIL)
     (GO A) B (SETQ V (CONS (CAR P) V)) (SETQ P (CDR P))
(COND((NOT(ATOM(CAR P)))(GO B))
((NOT(PEQ(CAR P)S))(GO BB)))
(SETQ S(CAR P))
(GO B)
BB (SETQ P(CONS (APPLY (PFNGET V
) (LIST V) (ALIST)) P)) (SETQ V NIL) (GO D)))) (PFNGET (LAMBDA (X) (COND
((ATOM(CAR X))(COND((OR(NUMBERP(CAR X))(LITER(CAR X))
(OPCHAR(CAR X)))(QUOTE PFN5))
  (T (GET (CAR X) (QUOTE PFN))))) (T (PFNGET (CDR X)))))) (
PGRT (LAMBDA (X Y) (AND (NOT (NULL X)) (OR (NULL Y) (AND (NUMBERP X) (ME
MBER Y (QUOTE (LP PFORMREAD EASSIGN RP CMA EQN PLS NEG PRD DVD LBK
RBK PWR))))(AND(OR(LITER X)(OPCHAR X))
  (MEMBER Y (QUOTE (LP RP CMA EQN PLS NEG PRD DVD LBK RBK PWR
PFORMREAD EASSIGN))))(MEMBER
    Y (GET X (QUOTE PGRT)))))) (PEQ (LAMBDA (X Y) (MEMBER Y (GET X (QUO
TE PEQ))))) (APARSE1 (LAMBDA (X) (COND ((NULL X) NIL)
((AND(EQ(CAR X)(QUOTE EQUOTE))(EQ(CADR X)(QUOTE LP)))
(CONS(QUOTE DEQUOTE)(APARSE1(CDR X))))
((AND(EQUAL(CAR
X) (QUOTE LP)) (EQUAL (CADR X) (QUOTE NEG))) (CONS (CAR X) (CONS (QUOTE
THETA) (APARSE1 (CDDR X))))) (T (CONS (CAR X) (APARSE1 (CDR X))))))) (AP
ARSE2 (LAMBDA (X) (COND ((EQUAL (CAR X) (QUOTE ATOM)) (COND ((NUMBERP (C
ADR X)) (PFN4 (CDR X))) (T (PFN6 (CDR X))))) ((ATOM X)X)(T(MAPLIST
X (FUNCTION (LAMBDA (U) (APARSE2 (CAR U))))))))) (PFN1 (LAMBDA (X)
 (COND ((NULL (CDDR X)) (LIST (CAR X))) (T (CONS (CAR X) (PFN1 (CDDR X))
))))) (PFN2 (LAMBDA (X Y) (COND ((NULL (CDDR X)) (LIST (CAR X) Y)) (T (C
ONS (CAR X) (PFN2 (CDDR X) Y)))))) (PFN4 (LAMBDA (X) (COND ((NULL (CDR X
)) (CAR X)) ((EQUAL (CADR X) (QUOTE POINT)) (PLUS (CAR X) (PFN41 (REVERS
E (CDDR X))))) (T (PFN4 (CONS (PLUS (TIMES 10 (CAR X)) (CADR X)) (CDDR X
))))))) (PFN5 (LAMBDA (X) (COND ((NULL (CDR X)) (LIST (QUOTE ATOM) (CAR
X))) (T (CONS (QUOTE ATOM) (CONS (CAR X) (CDADR X))))))) (PFN6 (LAMBDA (
X) (PROG (U) (SETQ U X) (CLEARBUFF) A (COND ((NULL U) (GO B))) (PACK (CA
R U)) (SETQ U (CDR U)) (GO A) B (RETURN (INTERN (MKNAM)))))))
DEFLIST((
(RP(RP CMA EQN PLS NEG PRD DVD LBK PWR))
(PLS(RP CMA EQN PLS NEG RBK))
(EQN(RP CMA))
(NEG(RP CMA EQN PLS NEG RBK))
(PRD(RP CMA EQN PLS NEG PRD DVD RBK))
(POINT(RP CMA EQN PLS NEG PRD DVD RBK PWR))
(DVD(RP CMA EQN PLS NEG PRD DVD RBK))
(RBK(LP RP CMA  EQN PLS NEG PRD DVD RBK PWR))
(PWR(RP CMA EQN RBK PWR PLS PRD DVD NEG))
(PFORMREAD(RP CMA EQN PLS NEG PRD DVD RBK PWR))
(EQUOTE(LP RP CMA EQN PLS NEG PRD DVD LBK RBK PWR EASSIGN PFORMREAD))
)PGRT)
DEFLIST((
(LP(RP CMA THETA))
(CMA(RP CMA RBK))
(THETA(RP))
(LBK(CMA RBK))
(DEQUOTE(LP))
)PEQ)
```

```
DEFLIST((
(EQN BPFN)
(PLS BPFN)
(PWR BPFN)
(PRD BPFN)
(PFORMREAD BPFN)
(EASSIGN BPFN)
(NEG(LAMBDA(X)(LIST(QUOTE PLS)(CAR X)(LIST(QUOTE PRD)-1(CADDR X)))))
(DVD BPFN)
(LP(LAMBDA(X)(COND
((EQUAL(CADR X)(QUOTE THETA))(LIST(QUOTE PRD)-1(CADDR X)))
((EQUAL(CAAR X)(QUOTE ATOM))(CONS(CAR X)(PFN1(CDDR X))))
((EQ(CAAR X)(QUOTE EQUOTE))(CONS(QUOTE FUN)
(CONS(CAR X)(PFN1(CDDR X)))))
((EQUAL(CADR X)(QUOTE LP))
(APPEND(REVERSE(CDR(REVERSE(CAR X))))(LIST
((LAMBDA(U)(COND
((EQ(CAR(LAST(CAR X)))(QUOTE EQUOTE))(CONS(QUOTE FUN)U))
(T U)))(CONS(LAST
(CAR X))(PFN1(CDDR X)))))))
((CDDDR X)(CONS(QUOTE ESET)(PFN1(CDR X))))
(T(CADR X)))))
(LBK(LAMBDA(X)(CONS(QUOTE NAM)(PFN2(CDDR X)(CAR X)))))
(POINT PFN5)
(DEQUOTE(LAMBDA(X)
(COND
((EQ(CADDR X)(QUOTE THETA))(LIST(QUOTE EQUOTE)
(LIST(QUOTE PRD)-1(CADDDR X))))
((EQ(CADDDR X)(QUOTE CMA))(LIST(QUOTE EQUOTE)
(CONS(QUOTE ESET)(PFN1(CDDR X)))))
(T(LIST(QUOTE EQUOTE)(CADDR X)))
)))
(EQUOTE(LAMBDA(X)X))
)PFN)
DEFINE((
(LAST(LAMBDA(X)(COND
((NULL(CDR X))(CAR X))
(T(LAST(CDR X))))))
(BPFN(LAMBDA(X)(LIST(CADR X)(CAR X)(CADDR X))))))
DEFINE((
(APARSE3(LAMBDA(X)(COND
((NUMBERP X)X)
((ATOM X)(LIST(QUOTE FORM)X))
((EQ(CAR X)(QUOTE EQUOTE))X)
(T(CONS(CAR X)(MAPLIST(CDR X)(FUNCTION
(LAMBDA(U)(APARSE3(CAR U))))))) )))
))
```

```
DEFINE((
(APARSE(LAMBDA(X)(GETFILE(QUOTE APARSE)(LIST X))))
))
REMPROPS((PFN41 PFNGET PGRT PEQ APARSE1 APARSE3
APARSE2 PFN1 PFN2 PFN4 PFN5 PFN6 BPFN)EXPR)
REMPROPS((RP PLS EQN NEG PRD POINT BAR RBK PWR
PFORMREAD EQUOTE)PGRT)
REMPROPS((LP CMA THETA LBK DEQUOTE)PEQ)
REMPROPS((EQN PLS PWR PRD PFORMREAD EASSIGN NEG
DVD LP LBK POINT DEQUOTE EQUOTE)PFN)
```

```
DEFINE ((
(APOFF3 (LAMBDA (X Y) (COND ((NULL (CDDR X)) (CONS Y (CDR X))) (T (CONS
(CAR X) (APOFF3 (CDR X) Y))))))
(WHOLEFILEREAD (LAMBDA (X Y) (PROG (U V) (FILESEEK X Y) A (SETQ U (READ)
) (COND ((EQUAL U (QUOTE $EOF$))(RETURN (PROG2 (FILEENDRD X Y) (REVERSE
V))))) (SETQ V (CONS (WFR1 U) V,) (GO A))))
(FILESEND (LAMBDA (X Y) (PROG (U) (SETQ U (READF X Y)) A (COND ((NULL U)
 (RETURN NIL))) (PRINT (CAR U)) (SETQ U (CDR U)) (GO A))))
(EDISPLAY(LAMBDA(X)(COND
((FILEGONE X(QUOTE PFORM))
(CHAIN(SUBST X(QUOTE NAME)(QUOTE((SAVE G001 T)(RESUME
LISPP $$A$$A EDPY $$A(A NAME $$A)A STOP)
(RENAME LISP LSPOUT NAME DFORM)(PRINT NAME DFORM)
(RESUME G001))))))
(T(CHAIN(SUBST X(QUOTE NAME)(QUOTE((SAVE G001 T)
(PRINT NAME DFORM)(RESUME G001)))))) )))
(ARGLIST (LAMBDA (Y) (COND ((ATOM (NTOL Y)) NIL) ((NOT (MEMBER (QUOTE PE
RM) (CAR (NTOL Y)))) (CDR (NTOL Y))) ((AND (EQUAL (CAR (NTOL Y)) (QUOTE
NAM)) (NOT (ATOM (NTOL (NTOL Y)))) (NOT (MEMBER (QUOTE PERM) (CAR (NTOL
(NTOL Y)))))) (CCR (NTOL (NTOL Y)))) (T NIL))))
(APOFF1 (LAMBDA (X Y) (COND ((NULL (CDDR X)) X) (T (CONS (COND ((NUMBERP
 (CADAR X)) (APOFF2 Y (CADAR X))) (T (CAR X))) (CONS (CADR X) (APOFF1 (C
DDR X) Y)))))))
(APOFF2 (LAMBDA (Y N) (COND ((ONEP N) (CAR Y)) (T (APOFF2 (CDR Y) (SUB1
N))))))
(READF (LAMBDA(Y X)(PROG(U)
(COND((FILEGONE Y X)(ERROR(LIST Y X(QUOTE UNDEFINED)))))
(FILESEEK Y X)(SETQ U(READ))(FILEENDRD
 Y X) (RETURN U))))
(APOFF (LAMBDA (X) (COND ((ATOM X) X) ((EQ (CAR X) (QUOTE DVD)) (EPRD (A
POFF (CADR X)) (EPWR (APOFF (CADDR X)) -1))) ((EQ (CAR X) (QUOTE NEG)) (
EPRD -1 (APOFF (CADR X)))) ((EQUAL (CAR X) (QUOTE FUNCTION)) (COND ((EQU
AL (CAADR X) (QUOTE NAM)) (APOFF (APOFF3 (CADR X) (CONS (CAR X) (CONS (N
TOL (CADR X)) (CDDR X)))))) (T (APOFF (CONS (CADADR X) (CDDR X)))))) ((E
QUAL (CAR X) (QUOTE TITLE)) (APOFF (CADDR X))) ((AND (EQUAL (CAR X) (QUO
TE DRV)) (ARGLIST X)) (CONS (CAR X) (MAPL (APOFF1 (CDR X) (CDR (ARGLIST
X))) (FUNCTION APOFF)))) ((OR (EQUAL (CAR X) (QUOTE ATOM)) (EQUAL (CAR X
) (QUOTE PAREN))) (APOFF (CADR X))) (T (CONS (CAR X) (MAPL (CDR X) (FUNC
TION APOFF))))))))
(EDELETE(LAMBDA(X)(PROG NIL
(FILEDELETE X(QUOTE FORM))
(FILEDELETE X(QUOTE PFORM))
(FILEDELETE X (QUOTE DFORM))
)))
(GETSUB (LAMBDA (EXP N) (CAR (GETSUB1 EXP))))
(GETSUB1(LAMBDA(EXP)(COND((ZEROP N)(PROG2(GETSUB3 EXP)(LIST EXP)))
((OR(ATOM EXP)(EQUAL
 (CAR EXP) (QUOTE ATOM))) (PROG2 (SETQ N (SUB1 N)) NIL)) (T (GETSUB2 (
PROG2 (SETQ N (SUB1 N)) (CDR EXP))))))))
(GETSUB2 (LAMBDA (EXP) (PROG (U) (COND ((NULL (CDR EXP)) (RETURN NIL)) (
(SETQ U (GETSUB1 (CAR EXP))) (RETURN U)) (T (RETURN (GETSUB2 (CDR EXP)))
)))))
(EPRINT(LAMBDA(X)(EPRINT1(RPLST(READF X(QUOTE FORM))))))
(EMIN (LAMBDA (Y) (COND ((NULL (CDDR Y)) (CAR Y)) ((LSSP (CAR Y) (CADR Y
)) (EMIN (CONS (CAR Y) (CDDR Y)))) (T (EMIN (CDR Y))))))
(LSSP (LAMBDA (X Y) (GRTP Y X)))
(GRTP (LAMBDA (X Y) (COND ((EQUAL X Y) NIL) ((AND (EQUAL X (QUOTE INF))
(NOT (EQUAL Y (QUOTE INF)))) T) ((EQUAL Y (QUOTE INF)) NIL) ((AND (NUMBE
RP X) (NUMBERP Y)) (GREATERP X Y)) (T (ERROR (QUOTE GRTP))))))
(EMINUSP (LAMBDA (X) (OR (AND (NUMBERP X) (MINUSP X)) (AND (EQUAL (CAR X
) (QUOTE FRT)) (MINUSP (CADDR X))))))
(PWRNUM (LAMBDA (X Y) (COND
((AND(EZEROP X)(MINUSP Y))(QUOTE INF))
((MINUSP Y)(PWRNUM(DVDNUM 1 X)(MINUS Y)))
((EQUAL (CAR X) (QUOTE FRT)) (DVDNUM (PWRNUM
 (CADR X) Y) (PWRNUM (CADDR X) Y)))  ((MINUSP X) (COND ((EODDP Y) (MINUS
(EXPT (MINUS X) Y))) (T (EXPT (MINUS X) Y)))) (T (EXPT X Y)))))
(EODDP (LAMBDA (X) (AND (NUMBERP X) (ONEP (REMAINDER X 2)))))
(QUESTION (LAMBDA (X Y) (PROG (U V W) (SETQ V X) (SETQ W Y) (PRINT (QUOT
E QUESTION)) B (COND ((NULL V) (GO A))) (SETQ U (INTERN (GET (GENSYM) (Q
UOTE PNAME))))(EASSIGN U(CAR V))(EDISPLAY
U) (SETQ W (SUBST U (CAR V) W)) (SETQ V (CDR V)) (GO B) A (PRINT W) C (S
ETQ U (ERSETQ (RDFLX))) (COND ((NULL U) (GO C)) ((EQUAL (CAR U) (QUOTE G
IVEUP)) (ERROR (QUOTE (RETURNING UPWARD)))) ((EQUAL (CAR U) (QUOTE ANSWE
RQ)) (GO D)) ((EQUAL (CAR U) (QUOTE ANSWER)) (GO E))) (SETQ V (ERSETQ (R
DFLX))) (COND ((NULL V) (GO C))) (ERSETQ (PRINT (EVALQUOTE (CAR V) (CAR
V)))) (PRINT (QUOTE (THIS QUESTION TO YOU IS PENDING))) (PRINT W) (GO C)
 D (SETQ V (ERSETQ (RDFLX))) (COND ((NULL V) (GO C))) (RETURN (CAR V)) E
 (SETQ U (ERSETQ (RDFLX))) (COND ((NULL U) (GO C))) (SETQ V (ERSETQ (RDF
LX))) (COND ((NULL V) (GO C))) (SETQ W (ERSETQ (EVALQUOTE (CAR U) (CAR V
)))) (COND ((NULL W) (GO C))) (RETURN W))))
(NUMBER (LAMBDA (X) (LENGTH X)))
))
DEFINE((
(WFR1(LAMBDA(X)(COND
((NULL X)NIL)
((ATOM X)(PRINT(LIST X)))
((ATOM(CAR X))(CONS(CAR X)(WFR1(CDR X))))
(T(ERROR X)))))
))
```

```
SPECIAL((N))
COMMON((DOLLAR))
COMMON((RANDOMNUMBER))
COMPILE((WHOLEFILEREAD FILESEND
PAGEUPDATE EDISPLAY ARGLIST APOFF1 APOFF2 READF APOFF
EDELETE  GETSUB GETSUB1 GETSUB2
 .EPRINT EMIN LSSP GRTP EMINUSP
PWRNUM EODDP QUESTION NUMBER ))
```

```
DEFINE
(((ENLOG (LAMBDA(X)(COND
((EONEP X)0)
((EZEROP X)(QUOTE(PRD -1 INF NIL)))
((EQUAL X(QUOTE INF))
 (QUOTE INF)) ((AND (EQUAL (CAR X) (QUOTE PWR)) (EQUAL (CADR X) (
QUOTE E))) (CADDR X)) (T (LIST (QUOTE NLOG) X NIL))))) (ENEG (LAMBDA (X)
 (EPRD X -1))) (EDVD (LAMBDA (X Y) (EPRD X (EPWR Y -1)))) (DEPEND (LAMBD
A (Y X) (COND ((ATOM Y) (EQUAL X Y))
(T(MEMBER(COND((EQ(CAR X)(QUOTE NAM))(RPLST
(COND((ATOM(NTOL X))X)
((AND(NULL(GET(CAR Y)(QUOTE PERM)))(EQ X(CAR Y)))T)
(T(NTOLSUBST X(CAR(NTOL X))))))))
(T X))(LAST Y))) )))
(EDRV(LAMBDA(X Y)(COND((EQUAL(CAR
   Y) (QUOTE DRV)) (CONS (CAR Y) (EDRV1 X (CDR Y)))) (T (APPEND (CONS (Q
UOTE DRV) X) (LIST Y NIL)))))) (EDRV1 (LAMBDA (X Y) (COND ((NULL (CDDR X
)) (EDRV2 (CAR X) (CADR X) Y)) (T (EDRV1 (CDDR X) (EDRV2 (CAR X) (CADR X
) Y))))))) (EDRV2 (LAMBDA (X Y Z) (COND ((NULL (CDDR Z)) (CONS X (CONS Y
Z))) ((EQUAL (CAR Z) X) (CONS X (CONS (PLUS Y (CADR Z)) (CDDR Z)))) (T (
CONS (CAR Z) (CONS (CADR Z) (EDRV2 X Y (CDDR Z)))))))) (NTOL (LAMBDA (X)
 (COND ((NULL (CDDR X)) (CAR X)) (T (NTOL (CDR X)))))) (FLOAT (LAMBDA (X
) (COND ((NUMBERP X) (PLUS X  0.0)) (T (QUOTIENT (PLUS (CADR X)  0.0) (P
LUS (CADDR X)  0.0)))))) (EPWR (LAMBDA (X Y) (COND ((EONEP Y) X) ((OR (
NULL X) (NULL Y))(LIST(QUOTE PWR X Y NIL)))
 ((AND (NOT (EQUAL X (QUOTE INF'))) (EZEROP Y)) 1) (
(EQUAL X(QUOTE INF)) (QUOTE INF))
((AND(EQ(CAR X)(QUOTE FRT))(LESSP(ABS(CADR X))(ABS(CADDR X))))
(EPWR(DVDNUM 1 X)(EPRD -1 Y)))
 ((AND (EENUMBERP X) (NUMBERP Y) (FIXP Y))
 (PWRNUM X Y)) ((AND (NUMBERP X) (NUMBERP Y) (NOT (MINUSP X))) (EXPT X Y
)) (T (LIST (QUOTE PWR) X Y NIL))))) (APND (LAMBDA (X Y) (COND ((NULL (C
DR X)) Y) (T (CONS (CA` X) (APND (CDR X) Y)))))) (MAPL (LAMBDA (X Y) (CO
ND ((NULL (CDR X)) X) (T (CONS (Y (CAR X)) (MAPL (CDR X) Y)))))) (PLSNUM
 (LAMBDA (X Y) (PROG (U) (SETQ U (DONUM X Y)) (RETURN (REDNUM (PLUS (TIM
ES (CAR U) (CADDDR U)) (TIMES (CADR U) (CADDR U))) (TIMES (CADR U) (CADD
DR U))))))) (PRDNUM (LAMBDA (X `) (PROG (U) (SETQ U (DONUM X Y)) (RETURN
 (REDNUM (TIMES (CAR U) (CADDR U`) (TIMES (CADR U) (CADDDR U))))))) (DVD
NUM (LAMBDA (X Y) (PROG (U) (SETQ U (DONUM X Y)) (RETURN (REDNUM (TIMES
(CAR U) (CADDDR U)) (TIMES (CADR U) (CADDR U))))))) (DONUM (LAMBDA (X Y)
 (PROG (U) (COND ((ATOM Y) (SET( U (CONS Y (CONS 1 U)))) (T (SETQ U (CON
S (CADR Y) (CONS (CADDR Y) U))));) (COND ((ATOM X) (SETQ U (CONS X (CONS
1 U)))) (T (SETQ U (CONS (CADR X) (CONS (CADDR X) U))))) (RETURN U)))) (
GCD (LAMBDA (X Y) (COND ((ZEROP (REMAINDER X Y)) Y) (T (GCD Y (REMAINDER
 X Y)))))) (REDNUM (LAMBDA (X Y) (COND
((ZEROP Y)(QUOTE INF))
((OR (FLOATP X)(FLOATP Y)) (QUOTIENT
     X Y)) ((ZEROP (REMAINDER X Y)) (QUOTIENT X Y)) (T (LIST (QUOTE FRT)
 (QUOTIENT X (GCD X Y)) (QUOTIENT Y (GCD X Y)) NIL))))) (EENUMBERP (LAMB
DA (X) (OR (NUMBERP X) (EQUAL (CAR X) (QUOTE FRT)))))) (STRIKE (LAMBDA (X
 Y) (COND ((NULL X) NIL) ((Y (CAR X)) (STRIKE (CDR X) Y)) (T (CONS (CAR
X) (STRIKE (CDR X) Y)))))) (EZEROP (LAMBDA (X) (AND (NUMBERP X) (ZEROP X
)))) (EONEP (LAMBDA (X) (OR (AND`(NUMBERP X) (EQUAL  0.1E1 (FLOAT X))) (
AND (EQUAL (CAR X) (QUOTE FRT)) (EQUAL (CADR X) (CADDR X)))))) (FTLEXP (
LAMBDA (X) (COND ((ZEROP X) 1) (T (TIMES X (FTLEXP (SUB1 X))))))) (UNION
LIST (LAMBDA (X Y) (COND ((NULL X) Y) ((MEMBER (CAR X) Y) (UNIONLIST (CD
R X) Y)) (T (CONS (CAR X) (UNIONLIST (CDR X) Y)))))) (MKPRD (LAMBDA (X)
(COND ((NULL (CDR X)) 1) ((NULL (CDDR X)) (CAR X)) (T (CONS (QUOTE PRD)
X))))) (MKPLS (LAMBDA (X) (COND ((NULL (CDR X)) 0) ((NULL (CDDR X)) (CAR
 X)) (T (CONS (QUOTE PLS) X))))) (FSIN (LAMBDA (X) (FPRD 1532106600 (FPL
S (FPWR 8364320344 (SPRD 552573(17) X)) (FPRD 8589949372 (FPWR 836432034
4 (SPRD 3064213199 X))))))) (FCOS (LAMBDA (X) (FPRD 4294974687 (FPLS (FP
WR 8364320344 (SPRD 5525736173 X)) (FPWR 8364320344 (SPRD 3064213199 X))
)))) (FTAN (LAMBDA (X) (PROG (U) (SETQ U (FPWR 8364320344 (SPRD 61284263
98 X))) (RETURN (FPRD 3064213200 (FDVD (FPLS 1 (FPRD 8589949372 U)) (FPL
S 1 U))))))) (FSEC (LAMBDA (X) (FDVD 1 (FCOS X)))) (FCSC (LAMBDA (X) (FD
VD 1 (FSIN X)))) (FCOT (LAMBDA (X) (FDVD 1 (FTAN X)))) (FSINH (LAMBDA (X
) (FPRD 3064213200 (FSIN (SPRD 5525736173 X))))) (FCOSH (LAMBDA (X) (FCO
S (SPRD 5525736173 X)))) (FTANH (LAMBDA (X) (FDVD (FSINH X) (FCOSH X))))
 (FCOTH (LAMBDA (X) (FDVD 1 (FTANH X)))) (FSECH (LAMBDA (X) (FDVD 1 (FCO
SH X)))) (FCSCH (LAMBDA (X) (FDVD 1 (FSINH X)))) (FDVD (LAMBDA (X Y) (FP
RD X (FPWR Y 8589949371)))) (GETC (LAMBDA (X) (PROG (U) (COND ((NUMBERP
X) (GO A)) ((EQUAL (CAR X) (QUOTE FRT)) (RETURN (ADVD (GETC (CADR X))
(GETC (CADDR X) ))))) (SETQ U (GET X (QUOTE CODE))) (COND ((NULL U) (GO
 B))) (RETURN U) B (SETQ U (SPLS 0 (RANNO))) (DEFLIST (LIST (LIST X U))
(QUOTE CODE)) (RETURN U) A (COND ((FLOATP X) (RETURN (MKCD X))) ((MINUSP
 X) (RETURN (ANEG (MINUS X))))) (RETURN (APLS 0 X))))) (FGFN (LAMBDA (X
Y) (PROG (U V) (SETQ U (CDR Y)) (SETQ V (FPWR (FPLS X (CAAR Y)) 12578605
91)) A (COND ((NULL (CDR U)) (RETURN V))) (SETQ V (FPWR (FPLS V (CAAR U))
 1257860591)) (SETQ U (CDR U)) (GO A)))) (MKCD (LAMBDA (X) (PROG (U V W)
(SETQ W (COND ((MINUSP X) (MINUS X)) (T X))) (SETQ U (LEFTSHIFT (LEFTSHI
FT W 8) -8)) (SETQ V (PLUS (LEFTSHIFT X -27) -155)) (COND ((MINUSP V) (S
ETQ W (ADVD U (APWR 2 (MINUS V))))) (T (SETQ W (APRD U (APWR 2 V))))) (C
OND ((MINUSP X) (RETURN (ANEG W))) (T (RETURN W)))))) (RANNO (LAMBDA NIL
 (PROG NIL (SETQ RANDOMNUMBER (TIMES 2187 RANDOMNUMBER)) (RETURN RANDOM
NUMBER)))) (ECODE (LAMBDA (X SIMPLEVEL) (COND ((OR (ATOM X) (EENUMBERP X
)) (GETC X)) ((NOT (NULL (GET (CAR X) (QUOTE NCODE)))) (SIMP32 (GET (CAR
 X) (QUOTE NCODE)) (MAPL (CDR X) (FUNCTION (LAMBDA (U) (ECODE U SIMPLEVE
L)))))) ((NOT (NULL (GET (CAR X) (QUOTE FCODE)))) (SIMP32 (GET (CAR X) (
QUOTE FCODE)) (ECODE (CADR X) (LOGXOR SIMPLEVEL 1)))) ((EQUAL (CAR X) (Q
UOTE PLS)) (ECODE1 (FUNCTION APLS) (MAPL (CDR X) (FUNCTION (LAMBDA (X) (
ECODE X SIMPLEVEL)))))) ((EQUAL (CAR X) (QUOTE PRD)) (ECODE1 (FUNCTION A
PRD) (MAPL (CDR X) (FUNCTION (LAMBDA (X) (ECODE X SIMPLEVEL)))))) ((EQUA
L (CAR X) (QUOTE PWR)) (APWR (ECODE (CADR X) SIMPLEVEL) (ECODE (CADDR X)
 (LOGXOR SIMPLEVEL 1)))) (T (FGFN (GETC (CAR X)) (MAPL (CDR X) (FUNCTION
 (LAMBDA (U) (CONS (ECODE U SIMPLEVEL) U))))))))) (SIMP32 (LAMBDA (GARG
Y) (GARG Y))) (ANEG (LAMBDA (X) (COND ((ZEROP SIMPLEVEL) (FPRD X 8589949
372)) (T (SPRD X 8589949371))))) (APRD (LAMBDA (X Y) (COND ((ZEROP SIMPL
EVEL) (FPRD X Y)) (T (SPRD X Y))))) (APLS (LAMBDA (X Y) (COND ((ZEROP SI
MPLEVEL) (FPLS X Y)) (T (SPLS X Y))))) (APWR (LAMBDA (X Y) (COND ((AND (
EQUAL SIMPLEVEL 1) (EQUAL Y 8589949372)) (SDVD 1 X)) ((ZEROP SIMPLEVEL)
(FPWR X Y)) (T (SPWR X Y))))) (ADVD (LAMBDA (X Y) (COND ((ZEROP SIMPLEVE
L) (FDVD X Y)) (T (SDVD X Y))))) (DPLS (LAMBDA (X Y) (DFUN X Y (FUNCTION
 APLS) (FUNCTION EPLS)))) (DFUN (LAMBDA (X Y AFUN EEFUN) (PROG(U V)(SETQ
 U(AFUN (CAR X) (CAR Y)))
(SETQ V (EEFUN(CDR X)(CDR Y)))
(COND((NULL V)(RETURN(CONS(ECODE V SIMPLEVEL)V)))
((OR(ZEROP U)(ONEP U))(RETURN (CONS
U U))) (RETURN (CONS U V))))) (DPRD (LAMBDA (X Y)
 (DFUN X Y (FUNCTION APRD) (FUNCTION EPRD)))) (DPWR (LAMBDA (X Y)
(COND((ZEROP(CAR X))(CONS(ECODE(EPWR(CDR X)(CDR Y)))
(EPWR(CDR X)(CDR Y))))(T (DFUN
X Y (FUNCTION APWR) (FUNCTION EPWR))))))(ECODE1 (LAMBDA (FN X) (COND((NU
LL (CDDR X)) (CAR X)) (T (FN (CAR X) (ECODE1 FN (CDR X))))))) (EFUN (LAM
BDA (X Y W NUMFUN Z NUMP) (PROG (U) (COND ((OR (NULL X) (NULL Y)) (RETUR
N (LIST W X Y NIL)
      )) ((NUMP X) (RETURN Y)) ((NUMP Y) (RETURN X)) ((OR (EQUAL X (QUOTE
 INF)) (EQUAL Y (QUOTE INF))) (RETURN (QUOTE INF)))) (SETQ U (EFUN1 X (E
FUN1 Y (CONS Z (LIST NIL))))) (COND ((NULL (CDDR U)) (RETURN (CAR U))) (
(NUMP (CAR U)) (COND ((NULL (CDDDR U)) (RETURN (CADR U))) (T (RETURN (CO
NS W (CDR U)))))) (T (RETURN (CONS W U))))))) (EFUN1 (LAMBDA (X Y) (PROG
 (U V) (SETQ U (CAR Y)) (SETQ V (CDR Y)) (COND ((EENUMBERP X) (SETQ U (N
UMFUN U X))) ((EQUAL (CAR X) W) (GO A)) (T (SETQ V (CONS X V)))) C (RETU
RN (CONS U V)) A (COND ((EENUMBERP (CADR X)) (GO B))) (SETQ V (APND (CDR
 X) V)) (GO C) B (SETQ U (NUMFUN U (CADR X))) (SETQ V (APND (CDDR X) V))
 (GO C)))) (EPLS (LAMBDA (X Y) (EFUN X Y (QUOTE PLS) (FUNCTION PLSNUM) 0
 (FUNCTION EZEROP)))) (EPRD (LAMBDA (X Y) (COND ((OR (EZEROP X) (EZEROP
Y))(COND((OR(EQ X(QUOTE INF))(EQ Y(QUOTE INF)))
(PROG2(PRINT(QUOTE INDETERMINATE))NIL) )(T O))
      ) (T (EFUN X Y (QUOTE PRD) (FUNCTION PRDNUM) 1 (FUNCTION EONEP)))))
) (RPLST (LAMBDA (X) (COND ((EQUAL (CAR X) (QUOTE ATOM)) (CADR X)) ((ATO
M X) X) ((NULL (CDR X)) NIL) (T (CONS (RPLST (CAR X)) (RPLST (CDR X))))
))
 (EQUOTE1    (LAMBDA (X) (COND ((NULL X) (LIST NIL)) ((ATOM X
) X) ((EQUAL (CAR X) (QUOTE NEG)) (LIST (QUOTE PRD) -1 (EQUOTE1 (CADR X
)) NIL)) ((EQUAL (CAR X) (QUOTE DVD)) (LIST (QUOTE PRD) (EQUOTE1 (CADR
X))(LIST (QUOTE PWR) (EQUOTE1    (CADDR X)) -1 NIL) NIL)) (T(CONS(EQUOTE1
 (CAR X)) (EQUOTE1    (CDR X)))))))
 (LAST (LAMBDA (X) (COND ((NULL (CDR X)) (CAR X)) (T (LAST (CDR X))))
))))
DEFLIST((
(EQUOTE(LAMBDA(X Y)(EQUOTE2(EQUOTE1(CAR X)))))
)FEXPR)
DEFINE((
(EQUOTE2(LAMBDA(X)(COND
((ATOM X)X)
((OR(EQ(CAR X)(QUOTE PLS))(EQ(CAR X)(QUOTE PRD)))(EQUOTE3 X(CAR X)))
(T(MAPL X(FUNCTION EQUOTE2))) )))
(EQUOTE3(LAMBDA(X Y)(COND
((NULL(CDR X))X)
((EQ(CAAR X)Y)(APND(CDR(EQUOTE2(CAR X)))(EQUOTE3(CDR X)Y)))
(T(CONS(EQUOTE2(CAR X))(EQUOTE3(CDR X)Y))) )))
(MKP(LAMBDA(X)(COND
((ATOM X)X)
(T(CONS(CAR X)(MKP1(MAPL(CDR X)(FUNCTION MKP))X NIL))) )))
(MKP1(LAMBDA(ARGLIST EXP PLIST)(COND
((NULL(CDR ARGLIST))(LIST PLIST))
(T(CONS(CAR ARGLIST)(MKP1(CDR ARGLIST)EXP
(UNIONLIST PLIST(COND
((AND(EQ(CAR EXP)(QUOTE NAM))(NULL(CDDR ARGLIST)))
(COND((ATOM(CAR ARGLIST))(LIST(RPLST EXP)))
(T(CONS(RPLST(NTOLSUBST EXP(CAAR ARGLIST)))(LAST(CAR ARGLIST))))))
((NUMBERP(CAR ARGLIST))NIL)
((ATOM(CAR ARGLIST))(LIST(CAR ARGLIST)))
((GET(CAAR ARGLIST)(QUOTE PERM))(LAST(CAR ARGLIST)))
(T(CONS(CAAR ARGLIST)(LAST(CAR ARGLIST))))) )))) )))
))
DEFINE((
(ABS(LAMBDA(N)(MAX N(MINUS N))))
))
DEFINE((
(ENUMBERP(LAMBDA(X)(OR(EQ X(QUOTE INF))(EENUMBERP X))))
))
```

```
COMMON((Y))
COMPILE((MAPL STRIKE))
UNCOMMON((Y))
COMMON((RANDOMNUMBER))
SPECIAL((W))
COMPILE((EFUN EFUN1))
UNSPECIAL((W))
COMPILE(( ENLOG ENEG EDVD DEPEND EDRV EDRV1 EDRV2
NTOL FLOAT EPWR APND PLSNUM PRDNUM DVDNUM DONUM GCD REDNUM
EENUMBERP EZEROP EONEP FTLEAP UNIONLIST
MKPRD MKPLS FSIN FCOS FTAN FSEC FCSC FCOT FSINH FCOSH
FTANH FCOTH FSECH FCSCH FDVD GETC FGFN
MKCD RANND ECODE SIMP32 ANEG APRD APLS APWR ADVD
DPLS DFUN DPRD DPWR ECODE1 EPLS EPRD RPLST EQUOTE EQUOTE1
 MKP MKP1  LAST))
```

```
DEFINE
((MKPRD (LAMBDA (X) (COND ((NULL (CDR X)) 1) ((NULL (CDDR X)) (CAR X))
(T (CONS (QUOTE PRD) X))))) (EFUN1 (LAMBDA (X Y) (PROG (U V) (SETQ U (CA
R Y)) (SETQ V (CDR Y)) (COND ((EENUMBERP X) (SETQ U (NUMFUN U X))) ((EQU
AL (CAR X) W) (GO A)) (T (SETQ V (CONS X V)))) C (RETURN (CONS U V)) A (
COND ((EENUMBERP (CADR X)) (GO B))) (SETQ V (APND (CDR X) V)) (GO C) B (
SETQ U (NUMFUN U (CADR X))) (SETQ V (APND (CDDR X) V)) (GO C)))) (APND (
LAMBDA (X Y) (COND ((NULL (CDR X)) Y) (T (CONS (CAR X) (APND (CDR X) Y))
)))) (EMINUSP (LAMBDA (X) (OR (AND (NUMBERP X) (MINUSP X)) (AND (EQUAL (
CAR X) (QUOTE FRT)) (MINUSP (CADDR X)))))) (NTOL (LAMBDA (X) (COND ((NUL
L (CDDR X)) (CAR X)) (T (NTOL (CDR X)))))) (FLOAT (LAMBDA (X) (COND ((NU
MBERP X) (PLUS X  0.0)) (T (QUOTIENT (PLUS (CADR X)  0.0) (PLUS (CADDR X
)  0.0))))))) (EDRV (LAMBDA (X Y) (COND ((EQUAL (CAR Y) (QUOTE DRV)) (CON
S (CAR Y) (EDRV1 X (CDR Y)))) (T (APPEND (CONS (QUOTE DRV) X) (LIST Y NI
L)))))) (EDRV1 (LAMBDA (X Y) (COND ((NULL (CDDR X)) (EDRV2 (CAR X) (CADR
 X) Y)) (T (EDRV1 (CDDR X) (EDRV2 (CAR X) (CADR X) Y)))))) (EDRV2 (LAMBD
A (X Y Z) (COND ((NULL (CDDR Z)) (CONS X (CONS Y Z))) ((EQUAL (CAR Z) X)
 (CONS X (CONS (PLUS Y (CADR Z)) (CDDR Z)))) (T (CONS (CAR Z) (CONS (CAD
R Z) (EDRV2 X Y (CDDR Z)))))))) (EENUMBERP (LAMBDA (X) (OR (NUMBERP X) (
EQUAL (CAR X) (QUOTE FRT))))) (UNIONLIST (LAMBDA (X Y) (COND ((NULL X) Y
) ((MEMBER (CAR X) Y) (UNIONLIST (CDR X) Y)) (T (CONS (CAR X) (UNIONLIST
 (CDR X) Y)))))) (MKP1 (LAMBDA (X) (CONS (CAR X) (MKP2 (MKP3 (CAR X)) (C
DR X))))) (MKP2 (LAMBDA (X Y) (COND ((NULL (CDR Y)) (LIST X)) (T (CONS (
CAR Y) (MKP2 (UNIONLIST X (COND ((EENUMBERP (CAR Y)) NIL) ((ATOM (CAR Y)
) (LIST (CAR Y))) (T (LAST (CAR Y))))) (CDR Y)))))))) (MKPLS (LAMBDA (X)
(COND ((NULL (CDR X)) 0) ((NULL (CDDR X)) (CAR X)) (T (CONS (QUOTE PLS)
X))))) (EPRD (LAMBDA (X Y) (COND ((OR (EZEROP X) (EZEROP Y)) 0) (T (EFUN
 X Y (QUOTE PRD) (FUNCTION PRDNUM) 1 (FUNCTION EONEP)))))) (EDVD (LAMBDA
 (X Y) (EPRD X (EPWR Y -1)))) (ENEG (LAMBDA (X) (EPRD X -1))) (EPLS (LAM
BDA (X Y) (EFUN X Y (QUOTE PLS) (FUNCTION PLSNUM) 0 (FUNCTION EZEROP))))
 (GRTP (LAMBDA (X Y) (COND ((EQUAL X Y) NIL) ((AND (EQUAL X (QUOTE INF))
 (NOT (EQUAL Y (QUOTE INF)))) T) ((EQUAL Y (QUOTE INF)) NIL) ((AND (NUMB
ERP X) (NUMBERP Y)) (GREATERP X Y)) (T (ERROR (QUOTE GRTP)))))) (LSSP (L
AMBDA (X Y) (GRTP Y X))) (FTLEXP (LAMBDA (X) (COND ((ZEROP X) 1) (T (TIM
ES X (FTLEXP (SUB1 X))))))) (EPWR (LAMBDA (X Y) (COND ((EONEP Y) X) ((OR
 (NULL X) (NULL Y)) NIL) ((AND (NOT (EQUAL X (QUOTE INF))) (EZEROP Y)) 1
) ((EQUAL X (QUOTE INF)) (QUOTE INF)) ((AND (EENUMBERP X) (EMINUSP Y)) (
EPWR (DVDNUM 1 X) (PRDNUM -1 Y))) ((AND (EENUMBERP X) (NUMBERP Y) (FIXP
Y)) (PWRNUM X Y)) ((AND (NUMBERP X) (NUMBERP Y) (NOT (MINUSP X))) (EXPT
X Y)) (T (LIST (QUOTE PWR) X Y NIL))))) (MKP (LAMBDA (X) (COND ((ATOM X)
 X) ((EQUAL (CAR X) (QUOTE NAM)) (CONS (CAR X) (MKP5 (CDR X) NIL))) (T (
MKP1 (MAPL X (FUNCTION MKP))))))) (MKP3 (LAMBDA (X) (COND ((EENUMBERP X)
 NIL) ((ATOM X) (COND ((NULL (GET X (QUOTE PERM))) (LIST (LIST X))) (T N
IL))) (T (LAST X))))) (MKP5 (LAMBDA (X Y) (COND ((NULL (CDDR X)) (COND (
(ATOM (CAR X)) (LIST (CAR X) (CONS (CAR X) Y))) (T (LIST (CONS (CAAR X)
(MKP2 NIL (MAPL (CDAR X) (FUNCTION MKP)))) (UNIONLIST (LAST (MKP2 NIL (M
APL (CDAR X) (FUNCTION MKP)))) (LIST (CONS (CAAR X) Y))))))) (T (CONS (C
AR X) (MKP5 (CDR X) (CONS (CAR X) Y))))))) (DEPEND (LAMBDA (Y X) (OR (EQ
UAL Y X) (MEMBER X (LAST Y)) (MEMBER (LIST X) (LAST Y))))) (EZEROP (LAMB
DA (X) (AND (NUMBERP X) (ZEROP X)))) (LAST (LAMBDA (X) (COND ((NULL (CDR
 X)) (CAR X)) (T (LAST (CDR X)))))) (MAPL (LAMBDA (X Y) (COND ((NULL (CD
R X)) X) (T (CONS (Y (CAR X)) (MAPL (CDR X) Y)))))) (EFUN (LAMBDA (X Y W
 NUMFUN Z NUMP) (PROG (U) (COND ((OR (NULL X) (NULL Y)) (RETURN NIL)) ((
NUMP X) (RETURN Y)) ((NUMP Y) (RETURN X)) ((OR (EQUAL X (QUOTE INF)) (EQ
UAL Y (QUOTE INF))) (RETURN (QUOTE INF)))) (SETQ U (EFUN1 X (EFUN1 Y (CO
NS Z (LIST NIL))))) (COND ((NULL (CDDR U)) (RETURN (CAR U))) ((NUMP (CAR
 U)) (COND ((NULL (CDDDR U)) (RETURN (CADR U))) (T (RETURN (CONS W (CDR
U)))))) (T (RETURN (CONS W U)))))) (EONEP (LAMBDA (X) (OR (AND (NUMBERP
 X) (EQUAL  0.1E1 (FLOAT X))) (AND (EQUAL (CAR X) (QUOTE FRT)) (EQUAL (C
ADR X) (CADDR X)))))) (PRDNUM (LAMBDA (X Y) (PROG (U) (SETQ U (DDNUM X Y
)) (RETURN (REDNUM (TIMES (CAR U) (CADDR U)) (TIMES (CADR U) (CADDDR U))
))))) (PLSNUM (LAMBDA (X Y) (PROG (U) (SETQ U (DDNUM X Y)) (RETURN (REDN
UM (PLUS (TIMES (CAR U) (CADDDR U)) (TIMES (CADR U) (CADDR U))) (TIMES (
CADR U) (CADDDR U))))))) (DVDNUM (LAMBDA (X Y) (PROG (U) (SETQ U (DDNUM
X Y)) (RETURN (REDNUM (TIMES (CAR U) (CADDDR U)) (TIMES (CADR U) (CADDR
U)))))))) (PWRNUM (LAMBDA (X Y) (COND ((EQUAL (CAR X) (QUOTE FRT)) (DVDNU
M (PWRNUM (CADR X) Y) (PWRNUM (CADDR X) Y))) ((MINUSP X) (COND ((ODDP Y
) (MINUS (EXPT (MINUS X) Y))) (T (EXPT (MINUS X) Y)))) (T (EXPT X Y)))))
 (DDNUM (LAMBDA (X Y) (PROG (U) (COND ((ATOM Y) (SETQ U (CONS Y (CONS 1
U)))) (T (SETQ U (CONS (CADR Y) (CONS (CADDR Y) U))))) (COND ((ATOM X) (
SETQ U (CONS X (CONS 1 U)))) (T (SETQ U (CONS (CADR X) (CONS (CADDR X) U
))))) (RETURN U)))) (REDNUM (LAMBDA (X Y) (COND ((OR (FLOATP X) (FLOATP
Y)) (QUOTIENT X Y)) ((ZEROP (REMAINDER X Y)) (QUOTIENT X Y)) (T (LIST (Q
UOTE FRT) (QUOTIENT X (GCD X Y)) (QUOTIENT Y (GCD X Y)) NIL))))) (GCD (L
AMBDA (X Y) (COND ((ZEROP (REMAINDER X Y)) Y) (T (GCD Y (REMAINDER X Y))
)))) (BILL NIL)))
```

```
COMMON((Y W NUMFUN NUMP))
COMPILE((MAPL EFUN EFUN1))
UNCOMMON((Y W NUMFUN NUMP))
COMPILE((MKPLS MKPRD EPRD EDVD ENEG  EPLS GRTP EMINUSP APND
LSSP FTLEXP EPWR MKP MKP1 MKP2 MKP3 MKP5 DEPEND
EZEROP LAST NTOL  EONEP PRDNUM PLSNUM DVDNUM
DDNUM REDNUM GCD EENUMBERP UNIONLIST FLOAT EDRV EDRV1 EDRV2))
STOP
```

```
DEFINE
(((COLLECT(LAMBDA(X Y)(CLCT3 X Y)))
  (RGCP (LAMBDA (X2 X1 Y2 Y1) (RGCP2 (ERSETQ (LIST (
RGCP1 X2 X1 Y2 Y1) (COND ((GRTP X1 Y1) (COND ((GRTP X2 Y1) NIL) ((LSSP X
2 Y2) (LIST Y2 Y1)) (T (LIST X2 Y1)))) ((LSSP X1 Y2) NIL) ((GRTP X2 Y2)
(LIST X2 X1)) (T (LIST Y2 X1)))) (RGCP1 Y2 Y1 X2 X1)))))) (RGCP1 (LAMBDA
(X2 X1 Y2 Y1) (COND ((GRTP X1 Y1) (LIST (LIST (COND ((GRTP X2 Y1) X2) (T
 (EPLS 1 Y1)))) X1) (COND ((LSSP X2 Y2) (LIST X2 (EPLS -1 Y2))) (T NIL)))
) ((LSSP X2 Y2) (LIST (LIST X2 (COND ((LSSP X1 Y2) X1) (T (EPLS Y2 -1))))
) NIL)) (T NIL)))) (RGCP2 (LAMBDA (X) (COND ((NULL X) NIL) (T (CAR X)))))
) (GRTP (LAMBDA (X Y) (COND ((AND (EQUAL X (QUOTE INF)) (NOT (EQUAL Y (Q
UOTE INF))))) T) ((EQUAL Y (QUOTE INF)) NIL) ((AND (NUMBERP X) (NUMBERP Y
)) (GREATERP X Y)) (T (ERROR (QUOTE GRTP))))))) (LSSP (LAMBDA (X Y) (GRTP
 Y X))) (CLCT411 (LAMBDA (X) (COND ((EQUAL X (QUOTE *I*)) 0) ((AND (EQUA
L (CAR X) (QUOTE PLS)) (NULL (CDDDDR X))) (COND ((AND (EQUAL (CADR X) (Q
UOTE *I*)) (NUMBERP (CADDR X))) (CADDR X)) ((AND (EQUAL (CADDR X) (QUOTE
 *I*)) (NUMBERP (CADR X))) (CADR X)) (T NIL))) (T NIL)))) (CLCT31 (LAMBD
A (X Y) (COND ((EQUAL (CAR X) (QUOTE EQN)) (LIST (QUOTE EQN) (CLCT31 (CA
DR X) Y) (CLCT31 (CADDR X) Y) NIL)) ((EQUAL (CAR X) (QUOTE PLS)) (CLCT32
 (CDR X) Y)) (T (CLCT32 (LIST X NIL) Y))))) (CLCT3 (LAMBDA (X Y) (CLCT31
(MKP(SIMPLIFY X))
(COND((EQ(CAR Y)(QUOTE ESET))
(REVERSE(CDR(REVERSE(CDR Y)))))
(T(LIST Y))))))
(CLCT32(LAMBDA(X Y)(CDR(SIMP2(MKP
 (MKPLS (CLCT9 (CLCT7 (CLCT8 X) (CLCT6 Y)) Y)) NIL 0))))) (CLCT6 (LAMBDA
 (Y) (COND ((NULL Y) (LIST 0)) (T (CONS (LIST (ECODE (CAR Y) 0)) (CLCT6
(CDR Y))))))) (CLCT7 (LAMBDA (X Y) (COND ((NULL X) Y) (T (CLCT7 (CDR X)
(CLCT1v (CAAR X) (CDAR X) Y (CDAR X) NIL)))))) (CLCT10 (LAMBDA (U X Y R
M) (COND ((NULL R) (COND ((NULL (CDDR Y)) (COND ((NULL M) (LIST (CAR Y)
(EPLS U (CADR Y)))) (T Y))) (T (CONS (CAR Y) (CLCT10 U X (CDR Y) X M))))
) ((EQUAL (CAAR Y) (CAAR R)) (COND ((NOT (NULL M)) (COND ((NULL (GET (QU
OTE CLCTVECTOR) (QUOTE TWOF))) (CLCT101 (QUESTION (LIST U) (CONS U (QUOT
E (CONTAINS MORE THAN ONE FACTOP RETURN YES IF YOU WISH TO COLLECT ON TH
E FIRST))))) Y)) (T Y))) (T (CLC)10 U X (CONS (CLCT11 (CAR R) (CAR Y)) (C
DR Y)) (CDR R) T)))) (T (CLCT10 U X Y (CDR R) M))))) (CLCT11 (LAMBDA (X
Y) (CONS (CAR Y) (CLCT111 NIL (CDR Y) (MKPRD (APND (CADDR X) (CADDDR X))
) (LIST (CADR X))))))) (CLCT111 (LAMBDA (X Y U Z) (PROG (V R) (COND ((NUL
L Y) (RETURN (CLCT1111 U Z))) ((NULL Z) (COND ((NULL X) (RETURN Y)) (T (
RETURN (CONS (CAR Y) (CONS (CADR Y) (CLCT111 NIL (CDDR Y) U X))))))) ((A
ND (NOT (ATOM (CAR Z))) (NULL (CDDAR Z))) (GO A)) ((AND (NOT (ATOM (CAR
Y))) (NULL (CDDAR Y))) (GO B)) ((EQUAL (ECODE (CAR Y) 0) (ECODE (CAR Z)
0)) (RETURN (CLCT111 X (CONS (CAR Y) (CONS (EPLS (CADR Y) U) (CDDR Y)))
U (CDR Z)))) (T (RETURN (CLCT111 (CONS (CAR Z) X) Y U (CDR Z))))) A (SET
Q V (CAR Z)) (COND ((AND (NOT (ATOM (CAR Y))) (NULL (CDDAR Y))) (SETQ R
(CAR Y))) (T (SETQ R (LIST (CAR Y) (CAR Y))))) (GO C) B (SETQ R (CAR Y))
 (COND ((AND (NOT (ATOM (CAR Z))) (NULL (CDDAR Z))) (SETQ V (CAR Z))) (T
 (SETQ R (LIST (CAR V) (CAR V))))) (GO C) C (SETQ V (RGCP (CAR V) (CADR
V) (CAR R) (CADR R))) (SETQ R (CADR Y)) (COND ((OR (NULL (CADR V)) (NULL
 V)) (GO E)) ((NULL (CAR V)) (GO D)) ((NOT (NULL (CAAR V))) (SETQ X (CON
S (CAAR V) X)))) (COND ((NOT (NULL (CADAR V))) (SETQ X (CONS (CADAR V) X
)))) D (COND ((NOT (NULL (CADR V))) (SETQ Y (CONS (CADR V) (CONS (EPLS (
CADR Y) U) (CDDR Y)))))) (COND ((NULL (CADDR V)) (GO G))) (COND ((NOT (N
ULL (CAADDR V))) (SETQ Y (CONS (CAADDR V) (CONS R Y))))) (COND ((NOT (NU
LL (CADADR (CDR V)))) (SETQ Y (CONS (CADADR (CDR V)) (CONS R Y))))) G (R
ETURN (CLCT111 X Y U (CDR Z))) E (SETQ X (CONS (CAR Z) X)) (GO G)))) (CL
CT1111 (LAMBDA (U Z) (COND ((NULL Z) NIL) (T (CONS (CAR Z) (CONS U (CLCT
1111 U (CDR Z)))))))) (CLCT101 (LAMBDA (V Y) (PROG NIL (COND ((EQUAL V (
QUOTE AYES)) (GO A)) ((EQUAL V (QUOTE YES)) (RETURN Y))) (ERROR (QUOTE H
ELP)) A (DEFLIST (LIST (LIST (QUOTE CLCTVECTOR) (QUOTE A))) (QUOTE TWOF)
) (RETURN Y)))) (CLCT9 (LAMBDA (X Y) (COND ((NULL Y) (LIST (CAR X) NIL))
```

```
 (T (APPEND (CLCT91 (CDAR X) (CAR Y)) (CLCT9 (CDR X) (CDR Y))))))) (CLCT
91 (LAMBDA (X Y) (COND ((NULL X) NIL) ((AND (NOT (ATOM (CAR X))) (NULL (
CDDAR X))) (CONS (COND ((EQUAL (CAAR X) (CADAR X)) (EPRD (CADR X) (EPWR
Y (CAAR X)))) (T (LIST (QUOTE SUM) (QUOTE *I*) (CAAR X) (CADAR X) (EPRD
(CADR X) (EPWR Y (QUOTE *I*))) NIL))) (CLCT91 (CDDR X) Y))) (T (CONS (EP
RD (CADR X) (EPWR Y (CAR X))) (CLCT91 (CDDR X) Y)))))) (CLCT4 (LAMBDA (Y
 V) (COND ((NULL (CDR Y)) NIL) ((EQUAL (CAAR Y) (QUOTE PWR)) (CONS (LIST
 (ECODE (CADAR Y) 0) (CADDAR Y) (CDR Y) V) (CLCT4 (CDR Y) (CONS (CAR Y)
V)))) ((EQUAL (CAAR Y) (QUOTE SUM)) (APPEND (CLCT41 (LIST NIL) (CADDAR Y
) (CADDDR (CAR Y)) (CLCT5 (SUBST (QUOTE *J*) (CADAR Y) (NTOL (CAR Y))))
(CDR Y) V) (CLCT4 (CDR Y) (CONS (CAR Y) V)))) (T (CONS (LIST (ECODE (CAR
 Y) 0) 1 (CDR Y) V) (CLCT4 (CDR Y) (CONS (CAR Y) V))))))) (CLCT41 (LAMBD
A (P R M U Y V) (COND ((NULL (CDR U)) NIL) ((AND (DEPEND (CAR U) (QUOTE
*I*)) (EQUAL (CAAR U) (QUOTE PWP)) (NOT (DEPEND (CADAR U) (QUOTE *I*))))
 (PROG (S Q) (SETQ S (CLCT411 (CADDAR U))) (COND ((NULL S) (SETQ Q (QUES
TION (LIST (CAR U)) (QUOTE (THE EXPONENT OF THIS FACTOR CAN NOT BE HANDL
ED RETURN YES IF YOU WISH TO CONTINUE WITHOUT COLLECTING IT)))))) (COND
((EQUAL Q (QUOTE YES)) (RETURN (CLCT41 (CONS (CAR U) P) R M (CDR U) Y V)
))) (SETQ Q (MKPRD (APND P (CDR U)))) (COND ((NOT (ZEROP S)) (SETQ Q (SU
BST (LIST (QUOTE PLS) (QUOTE *I*) (MINUS S) NIL) (QUOTE *I*) Q)))) (RETU
RN (CONS (LIST (ECODE (CADAR U) 0) (LIST (EPLS R S) (EPLS M S)) Y (CONS
Q V)) (CLCT41 (CONS (CAR U) P) R M (CDR U) Y V))))) ((NOT (DEPEND (CAR U
) (QUOTE *I*))) (CONS (LIST (ECODE (COND ((EQUAL (CAAR U) (QUOTE PWR)) (
CADAR U)) (T (CAR U))) 0) (COND ((EQUAL (CAAR U) (QUOTE PWR)) (CADDAR U)
) (T 1))) Y (CONS (LIST (QUOTE SUM) (QUOTE *J*) R M (SUBST (QUOTE *J*) (Q
UOTE *I*) (MKPRD (APND P (CDR U)))) NIL) V)) (CLCT41 (CONS (CAR U) P) R
M (CDR U) Y V))) (T (CLCT41 (CONS (CAR U) P) R M (CDR U) Y V))))) (CLCT8
 (LAMBDA (X) (COND ((NULL (CDR X)) NIL) (T (CONS (CONS (CAR X) (CLCT4 (C
LCT5 (CAR X)) (LIST NIL))) (CLCT8 (CDR X))))))) (CLCT5 (LAMBDA (X) (COND
 ((EQUAL (CAR X) (QUOTE PRD)) (CDR X)) (T (LIST X NIL)))))))
```

```
DEFINE((
(DELSUBST(LAMBDA(EXP OLDDEL NEWDELFREE)(SUBSTA
(QUOTE(DELSUBST1 EXP NEWDELFREE))
(LIST(QUOTE DRV)(CADR OLDDEL) 1(QUOTE DOLLAR)NIL)
EXP)))
(DELSUBST1(LAMBDA(WFREE DEL)(SUBSTA
(QUOTE(LIST(QUOTE DRV)(CADR EXP)1(NTOL WFREE)NIL))
(QUOTE(DEL DOLLAR NIL))DEL)))
))
```

```
DEFINE
(((DIFF (LAMBDA(Y X)(DIFF2 (MKP Y) X)))
(DIFF2(LAMBDA(Y X)(COND((NOT(
DEPEND Y X)) 0) ((ATOM Y) (COND ((EQUAL X Y) 1) (T 0))) ((NOT (NULL (GET
 (CAR Y) (QUOTE DIFF)))) (EPRD (SUBST (CADR Y) (QUOTE *Y*) (GET (CAR Y)
(QUOTE DIFF))) (DIFF2 (CADR Y) X))) ((NULL (GET (CAR Y) (QUOTE PERM))) (
COND ((EQUAL (CAR Y) X) 1) (T (MKPLS (DIFF8 (CDR Y) Y X))))) ((MEMBER (C
AR Y) (QUOTE (XST SUM))) (LIST (CAR Y) (CADR Y) (CADDR Y) (CADDDR Y) (DI
FF2 (CADR (CDDDR Y)) X) NIL)) (T (SELECT (CAR Y) ((QUOTE PLS) (MKPLS (DI
FF3 (CDR Y) X))) ((QUOTE PRD) (MKPLS (DIFF4 (CDR Y) X NIL))) ((QUOTE PWR
) (EPLS (EPRD (DIFF2 (CADR Y) X) (EPRD (CADDR Y) (EPWR (CADR Y) (EPLS (C
ADDR Y) -1))))) (EPRD Y (EPRD (ENLOG (CADR Y)) (DIFF2 (CADDR Y) X))))) ((
QUOTE ITG) (COND ((AND (EQUAL (CADR Y) X) (EQUAL (CADDR Y) (QUOTE IDF)))
 (CADR (CDDDR Y))) (T (EPLS (EPLS (EPRD (DIFF2 (CADDR Y) X) (ENEG (SUBST
 (CADDR Y) (CADR Y) (CADR (CDDDR Y))))) (EPRD (DIFF2 (CADDDR Y) X) (SUBS
T (CADDDR Y) (CADR Y) (CADR (CDDDR Y))))) (COND ((DEPEND (CADR (CDDDR Y)
) X) (LIST (CAR Y) (CADR Y) (CADDR Y) (CADDDR Y) (DIFF2 (CADR (CDDDR Y))
 X) NIL)) (T 0)))))) ((QUOTE DRV) (COND ((AND (EQUAL (CAR (NTOL Y)) (QUO
TE NAM)) (NOT (ATOM (NTOL (NTOL Y))))) (MKPLS (DIFF8 (CDR (NTOL (NTOL Y)
)) Y X))) ((OR (ATOM (NTOL Y)) (MEMBER (QUOTE PERM) (CAR (NTOL Y)))) (ED
RV (LIST X 1) Y)) (T (MKPLS (DIFF8 (CDR (NTOL Y)) Y X))))) ((QUOTE EQN)
(LIST (CAR Y) (DIFF2 (CADR Y) X) (DIFF2 (CADDR Y) X) NIL)) ((QUOTE STT)
(CONS (CAR Y) (MAPL (CDR Y) (FUNCTION (LAMBDA (U) (DIFF2 U X)))))) ((QUO
TE TMS) (CONS (QUOTE DRV) (CONS X (CONS 1 (LIST Y NIL))))) ((QUOTE CND)
(CONS (CAR Y) (DIFF6 (CDR Y) X))) ((QUOTE NAM) (COND ((EQUAL (RPLST X) (
RPLST Y)) 1) (T (MKPLS (DIFF8 (CDR (NTOL Y)) Y X))))) (ERROR (LIST Y X (
QUOTE DIFFERENTIAL NOT DEFINED))))))))) (DIFF3 (LAMBDA (Y X) (COND ((NULL
 (CDR Y)) Y) ((DEPEND (CAR Y) X) (CONS (DIFF2 (CAR Y) X) (DIFF3 (CDR Y)
X))) (T (DIFF3 (CDR Y) X))))) (DIFF4 (LAMBDA (Y X Z) (COND ((NULL (CDR Y
)) Y) ((DEPEND (CAR Y) X) (CONS (EPRD (DIFF2 (CAR Y) X) (MKPRD (APPEND Z
 (CDR Y)))) (DIFF4 (CDR Y) X (CONS (CAR Y) Z)))) (T (DIFF4 (CDR Y) X (CO
NS (CAR Y) Z)))))) (DIFF6 (LAMBDA (Y X) (COND ((NULL (CDR Y)) Y) (T (CON
S (LIST (CAAR Y) (DIFF2 (CADAR Y) X) NIL) (DIFF6 (CDR Y) X)))))) (DIFF8
(LAMBDA (Y Z X) (COND ((NULL (CDR Y)) Y) ((DEPEND (CAR Y) X) (CONS (EPRD
 (EDRV (LIST (CAR Y) 1) Z) (DIFF2 (CAR Y) X)) (DIFF8 (CDR Y) Z X))) (T (
DIFF8 (CDR Y) Z X))))))))
```

```
COMPILE((DIFF DIFF3 DIFF4 DIFF6 DIFF8))
STOP
        *EOF*
```

```
DEFINE
(((DRVDO(LAMBDA(X Y)(DIFFDO1 X Y)))
(DIFFDO1(LAMBDA(X Y)(COND((ATOM X)
X) ((EQUAL (CAR X) (QUOTE DRV)) (DIFFDO2 (CDR X) (DIFFDO1 (NTOL X) Y) Y
NIL)) (T (CONS (CAR X) (MAPL (CDR X) (FUNCTION (LAMBDA (U) (DIFFDO1 U Y)
))))))) (DIFFDO2 (LAMBDA (X Y Z W) (COND ((NULL (CDDR X)) (COND ((NULL
W) Y) (T (CONS (QUOTE DRV) (APPEND W (LIST Y NIL)))))) ((EQUAL (CAR X) Z
) (DIFFDO2 (CDDR X) (DIFFDO3 (CADR X) Y Z) Z W)) (T (DIFFDO2 (CDDR X) Y
Z (CONS (CAR X) (CONS (CADR X) W))))))) (DIFFDO3 (LAMBDA (N Y Z) (COND (
(ZEROP N) Y) (T (DIFFDO3 (SUB1 N) (MKP (DIF/ Y Z)) Z)))))))
DEFINE((
(DRVFACTOR(LAMBDA(EXP DRVFREE NFREE)(SUBSTA
(QUOTE(CONS(QUOTE DRV)(DRVFACTOL1(CDR EXP)DRVFREE
 NFREE)))
(QUOTE(DRV DOLLAR
((LAMBDA(U)(WITHRESPECTTO U DRVFREE)))))EXP)))
(DRVFACTOR1(LAMBDA(Y X N)(COND
((EQUAL(CAR Y)X)(COND((NOT(LESSP(CADR Y)N))
(COND((ONEP(CADR Y))(DRVFACTOR2(CDDR Y)X N))
(T(CONS(CAR Y)(CONS(DIFFERENCE(CADR Y)N)
(DRVFACTOR2(CDDR Y)X N))))))
(T Y)))
(T(CONS(CAR Y)(CONS(CADR Y)(DRVFACTOR1(CDDR Y)X N)))))))
(DRVFACTOR2(LAMBDA(Y X N)(COND
((NULL(CDDR Y))(CONS(LIST(QUOTE DRV)X N)(CAR Y)NIL)
(CDR Y)))
(T(CONS(CAR Y)(CONS(CADR Y)(DRVFACTOR2(CDDR Y)X N)))))))
))
```

```
DEFINE((
(EVALUATE(LAMBDA(X SET)(PROG(U MARKFREE V)
(SETQ U(EVALUATE4(SETTER SET)))
(SETQ V X)
A(SETQ MARKFREE NIL)
(SETQ V(EVALUATE1 V U))
(COND(MARKFREE(GO A)))
(RETURN V) )))
(EVALUATE1(LAMBDA(X Y)(COND
((NULL(CDR Y))X)
(T(EVALUATE1(EVALUATE2 X(CAR Y))(CDR Y))) )))
(EVALUATE2(LAMBDA(X Y)(PROG(U)
(COND((SETQ U(EVALUATE5 X(CADR Y)(CADDR Y)))
(PROG2(SETQ MARKFREE T)(RETURN U)))
((ATOM X)(RETURN X))
(T(RETURN(CONS(CAR X)(MAPL(CDR X)(FUNCTION(LAMBDA(V)
(EVALUATE2 V Y))))))))) )))
(EVALUATE5(LAMBDA(EXP FUN FORM)(COND
((NULL FORM)NIL)
((ATOM FUN)(COND((EQ EXP FUN)FORM)(T NIL)))
((EQ(CAR FUN)(QUOTE NAM))(COND
((EQ(CAR EXP)(QUOTE NAM))(EVALUATE5(NTOL EXP)(NTOL FUN)
(EVALUATE6(TWOOFF EXP)(TWOOFF FUN)FORM)))
(T NIL)))
((EQ(CAR FUN)EXP)FORM)
((EQ(CAR FUN)(CAR EXP))(EVALUATE6(LASTOFF EXP)(LASTOFF FUN)FORM))
(T NIL) )))
(LASTOFF(LAMBDA(X)(COND
((NULL(CDR X))NIL)
(T(CONS(CAR X)(LASTOFF(CDR X)))) )))
(EVALUATE6(LAMBDA(EXP FUN FORM)(COND
((NULL FUN)(COND((NULL EXP)FORM)(T NIL)))
((NULL EXP)NIL)
((EQUAL(CAR EXP)(CAR FUN))(EVALLATE6(CDR EXP)(CDR FUN)FORM))
((NUMBERP(CAR FUN))NIL)
(T(EVALUATE6(CDR EXP)(CDR FUN)(SUBST
(CAR EXP)(CAR FUN)FORM))) )))
(EVALUATE4(LAMBDA(X)(COND
((NULL(CDR X))X)
((AND(EQ(CAAR X)(QUOTE EQN))
(OR(ATOM(CADAR X))(EQ(CAADAR X)(QUOTE NAM)))
(NULL(GET(CAADAR X)(QUOTE PERM))))
(CONS(CAR X)(EVALUATE4(CDR X))))
(T(EVALUATE4(CDR X))) )))
))
```

```
DEFINE
(((EXPAND(LAMBDA(Y)(MULT2 Y)))
                        (MULT31 (LAMBDA (Y) (COND ((NULL (CDDR Y)) NIL) (T (CO
NS (CAR Y) (MULT31 (CDR Y))))))) (MULT2 (LAMBDA (Y) (COND ((EQUAL (CAR Y
) (QUOTE EQN)) (LIST (QUOTE EQN) (MULT2 (CADR Y)) (MULT2 (CADDR Y)) NIL)
) (T (CDR (SIMP2 (MKPLS (APPEND (MULT3 Y) (LIST NIL))) NIL 0))))))) (MULT
3 (LAMBDA (Y) (SELECT (CAR Y) ((QUOTE PLS) (MULT4 (MAPL (CDR Y) (FUNCTIO
N MULT3))))) ((QUOTE PRD) (MULT5 (MAPL (CDR Y) (FUNCTION MULT3)) NIL)) ((
QUOTE DRV) (MULT6 (MULT31 Y) (MULT3 (NTOL Y)))) ((QUOTE ITG) (MULT6 (MUL
T31 Y) (MULT3 (NTOL Y)))) ((QUOTE PWR) (MULT7 (MULT3 (CADR Y)) (MULT2 (C
ADDR Y)))) (LIST Y)))) (MULT4 (LAMBDA (Y) (COND ((NULL (CDDR Y)) (CAR Y)
) (T (MULT4 (CONS (APPEND (CAR Y) (CADR Y)) (CDDR Y))))))) (MULT5 (LAMBD
A (Y V) (COND ((NULL (CDR Y)) V) (T (MULT5 (CDR Y) (MULT51 (CAR Y) V NIL
))))))) (MULT51 (LAMBDA (U V Z) (COND ((NULL V) U) ((NULL U) Z) (T (MULT5
1 (CDR U) V (APPEND (MULT511 (CAR U) V) Z)))))) (MULT511 (LAMBDA (X Y) (
COND((NULL Y)NIL)(T(CONS(EPRD X(CAR Y))(MULT511 X(CDR Y))))
))) (
MULT6 (LAMBDA (X Y) (COND ((NULL Y) NIL) (T (CONS (APPEND X (CONS (CAR Y
) (LIST NIL))) (MULT6 X (CDR Y))))))) (MULT7 (LAMBDA (X Y) (COND ((NULL
(CDR X)) (LIST (EPWR (CAR X) Y))) ((AND (NUMBERP Y) (NOT (MINUSP
          Y ))) (MULT71 (LIST 1) X       Y )) (T (LIST (EPWR (MKPLS (APPE
ND X (LIST NIL)))        Y ))))))) (MULT71 (LAMBDA (Y X N) (COND ((ZEROP N)
 Y) (T (MULT71 (MULT51 X Y NIL) X (SUB1 N))))))))
```

```
DEFINE((
(FACTOROUT1(LAMBDA(X FACTOR NAME)(COND
((EQ(CAR X)(QUOTE EQN))(LIST(CAR X)
(FACTOROUT1(CADR X)FACTOR NAME)
(FACTOROUT1(CADDR X)FACTOR NAME)NIL))
((EZEROP X)X)
((EQ(CAR X)(QUOTE PLS))(EPRD NAME(MKPLS(MAPL(CDR X)
(FUNCTION(LAMBDA(U)(SIMPLIFY(COPY(EDVD U FACTOR))))))))))
(T(EPRD NAME(SIMPLIFY(COPY(EDVD X  FACTOR)))) )))
))
DEFLIST((
(FACTOROUT(LAMBDA(X Y)(EVAL(CONS
(QUOTE FACTOROUT1)(COND
((NULL(CDDR X))(CONS(CAR X)(CONS(CADR X)(CDR X))))
(T X)))
Y)))
)FEXPR)
```

```
DEFLIST((
(GROUP(LAMBDA(ARGS ALIST)(PROG(UFREE W FLAG)
(COND((CDR ARGS)(ERROR(QUOTE(WRONG NUMBER OF ARGUMENTS FOR
 GROUP)))))
(COND(INPFORM(SETQ FLAG T)))
(CSETQ INPNUM 0)
(CSETQ INPFORM(READF(EVAL(CAR(CADAR ARGS))ALIST)(QUOTE PFORM)))
(SETQ UFREE(EVAL(SUBST(QUOTE GETINSUB)(QUOTE PFORMREAD)(CAR ARGS))
ALIST))
(SETQ W INPFORM)
(COND((NULL FLAG)(CSETQ INPFORM NIL)))
(RETURN(CONS(GROUP1(CDR W)(CAR L))(CDR UFREE))) )))
)FEXPR)
DEFINE((
(GROUP1(LAMBDA(X OP)(PROG(U)(RETURN(COND
((NULL(CDR X))NIL)
((ATOM(CAR X))(GROUP1(CDR X)OP))
((EQ(CAAR(LAST(CAR X)))(QUOTE SUBMARK))(OP)
((SETQ U(GROUP1(CDAR X)(CAAR X)))U)
(T(GROUP1(CDR X)OP)))) )))
))
```

```
DEFINE((
(LIMIT(LAMBDA(EXP X N)(SIMPLIFY(LIMIT1(SIMPLIFY EXP)X N)) ))
(LIMIT1(LAMBDA(EXP X N)(COND
((EQUAL EXP X)N)
((ATOM EXP)EXP)
(T(SELECT(CAR EXP)
((QUOTE PRD)(LIMIT2(CDR EXP)X N))
(MAPL EXP(FUNCTION(LAMBDA(U)(LIMIT1 U X N))))))) ))
(LIMIT2(LAMBDA(Y X M)(PROG(U V W N D)
(SETQ N 1)
(SETQ D 1)
(SETQ W 1)
(SETQ U Y)
B(COND((NULL(CDR U))(GO A)))
(SETQ V(SIMPLIFY(LIMIT1(CAR U)X M)))
(COND((EZEROP V)(SETQ N(EPRD(CAR U)N)))
((EQ V(QUOTE INF))(SETQ D(EPRD(CAR U)D)))
(T(SETQ W(EPRD V W))))
(SETQ U(CDR U))
(GO B)
A(RETURN
(COND((EONEP D)(COND((EONEP N)W)(T 0)))
((EONEP N)(QUOTE INF))
(T(EPRD W(LIMIT(EDVD(DIFF N X)(DIFF(SIMPLIFY(EDVD 1(COPY D)))X))X M)))))
)))
))
```

```
LOAD((SIMP1))
EVALREAD(SIMP1 COMP NIL)
LOAD((SIMP2))
EVALREAD(SIMP2 COMP NIL)
LOAD((BASIC1))
EVALREAD(BASIC1 COMP NIL)
LOAD((BASIC2))
EVALREAD(BASIC2 COMP NIL)
LOAD((NBASIC))
EVALREAD(NBASIC COMP NIL)
LOAD((FIELD CHAIN SETUP))
CSET((INLIST NIL)
```

NBASIC LISP

```
DEFINE((
(EASSIGN(LAMBDA(NAME EXPRESSION)(PROG2
(EASS NAME EXPRESSION)
EXPRESSION)))
(EASS(LAMBDA(NAME EXP)(PROG NIL
(FILEDELETE NAME(QUOTE PFORM))
(FILEDELETE NAME (QUOTE DFORM))
(COND((AND(NOT(EQ NAME(QUOTE LAST)))
(NOT(FILEGONE NAME(QUOTE FORM))))
(FILEWRITE(QUOTE OLD)(QUOTE FORM)
(READF NAME(QUOTE FORM)))))
(FILEWRITE NAME (QUOTE FORM) EXP)
(RETURN NAME) )))
(EEQN(LAMBDA(X Y)(LIST(QUOTE EQN)X Y NIL)))
(EXCHANGE(LAMBDA(X)(COND
((CDDDDR X)X)
(T(LIST(CAR X)(CADDR X)(CADR X)(CADDDR X))))))
(MATCH(LAMBDA(EXP PATT CONST)(PROG(MALIST)
(SETQ MALIST(ALIST))
(COND((MATCH1 EXP PATT)(RETURN(EVAL CONST MALIST))))
(RETURN EXP) )))
(MATCH1(LAMBDA(EXP PATT)
(AND(MATCH2 EXP PATT)
(OR(NULL(LAST PATT))(APPLY(CAR(LAST PATT))
(LIST EXP)MALIST)))))
(SUBSTA(LAMBDA(CONST PATT EXP)(SUBSTA1 EXP)))
(SUBSTA1(LAMBDA(X)(COND
((ATOM X)(MATCH X PATT CONST))
(T(MATCH(CONS(CAR X)(MAPL(CDR X)(FUNCTION SUBSTA1)))
PATT CONST)) )))
(GETFILE(LAMBDA(OP ARGS)(PROG(U)
(EVALREAD OP(QUOTE LIST)NIL)
(SETQ U(COND
((GET OP(QUOTE FEXPR))(EVAL(CONS OP ARGS)(ALIST)))
(T(APPLY OP ARGS(ALIST)))))
(EVALREAD OP(QUOTE ERASE)NIL)
(CSETQ INLIST NIL)
(RETURN U))))
(DRVZERO(LAMBDA(EXP XFREE)(SUBSTA 0
(QUOTE(DRV DOLLAR((LAMBDA(U)(WITHRESPECTTO U XFREE)))))EXP
)))
(TERM(LAMBDA(X N)(COND
((NULL(CDR X))NIL)
((LESSP N 1)(CAR X))
(T(TERM(CDR X)(SUB1 N))) )))
(CENSUS(LAMBDA(X)(COND
((ATOM X)1)
((NULL(CDR X))0)
(T(PLUS(CENSUS(CAR X))(CENSUS(CDR X)))) )))
(LEFT(LAMBDA(X)(CADR X)))
(DEPENDENCE(LAMBDA(X)(COND
((EENUMBERP X)NIL)
((ATOM X)(ESET X))
(T(CONS(QUOTE ESET)(NILON(LAST(MKP X))))) )))
(RIGHT(LAMBDA(X)(CADDR X)))
(NEWNAME(LAMBDA()(PROG()
(CLEARBUFF)
(PACK(QUOTE F))
(NEWNAME1(EXPLODE(CSETQ NAMENUMBER(ADD1 NAMENUMBER))))
(RETURN(INTERN(MKNAM))))))
(NEWNAME1(LAMBDA(X)(COND
((NULL X)NIL)
(T(PROG2(PACK(CAR X))
(NEWNAME1(CDR X))))))))
(MATCH2(LAMBDA(EXP PATT)(COND
((NULL(CDR PATT))(NULL(CDR EXP)))
((EQ(CAR PATT)(QUOTE DOLLAR))T)
((ATOM EXP)NIL)
((NULL(CDR EXP))NIL)
((EQUAL(CAR EXP)(CAR PATT))(MATCH2(CDR EXP)
(CDR PATT)))
(T NIL))))
(WITHRESPECTTO(LAMBDA(EXP X)(MEMBER X(TWOOFF EXP))))
(NTOLSUBST(LAMBDA(X Y)(COND
((NULL(CDDR X))(CONS Y(CDR X)))
(T(CONS(CAR X)(NTOLSUBST(CDR X)Y))) )))
(REPLACESUB(LAMBDA(X)(SUBSTA
(QUOTE(CAR(LAST EXP)))
(QUOTE(DOLLAR))(LAMBDA(U)(EQ(CAAR(LAST U))
(QUOTE SUBMARK))))))(X)))
(GETSUB3(LAMBDA(X)(COND
((NULL(CDR X))(RPLACA X(LIST(LIST(QUOTE SUBMARK)(CSETQ INPNUM
(ADD1 INPNUM))(NIL)))))
(T(GETSUB3(CDR X))) )))
(TWOOFF(LAMBDA(X)(COND
((NULL(CDDR X))NIL)
(T(CONS(CAR X)(TWOOFF(CDR X))))))))
))
CSET(NAMENUMBER 1)
DEFINE((
(PUT(LAMBDA(X Y Z)(DEFLIST(LIST
(LIST Y X))Z))))
DEFLIST((
(ALIST(LAMBDA(X Y)Y))
(NAM(LAMBDA(X Y)(CONS(QUOTE NAM)(NILON(EVLIS X Y)))))
(DRV(LAMBDA(X Y)(DRV1(EVLIS X Y)(EVAL(LAST X)Y)))))
(FUN(LAMBDA(X Y)(NILON(EVLIS X Y))))
(ESET(LAMBDA(X Y)(CONS(QUOTE ESET)(NILON(EVLIS X Y)))))
)FEXPR)
CSET(INPNUM 1)
DEFINE((
(DRVFACTOR(LAMBDA(X Y Z)(GETFILF(QUOTE DRVFACTOR)
(LIST X Y Z))))
(DELSUBST(LAMBDA(X Y Z)(GETFILE(QUOTE DELSUBST)(LIST X Y Z))))
(SUMEACH(LAMBDA(X)(GETFILE(QUOTE SUMEACH)(LIST X))))
(SUMEXPAND(LAMBDA(X)(GETFILE(QULTE SUMEXPAND)(LIST X))))
(MULTIPLYTHROUGH(LAMBDA(X Y)(GETFILE(QUOTE MULTIPLYTHROUGH)
(LIST X Y))))
(DIFF(LAMBDA(X Y)(GETFILE(QUOTE DIFF)(LIST X Y))))
(DRVDO(LAMBDA(X Y)(GETFILE(QUOTI DRVDO)(LIST X Y))))
(EXPAND(LAMBDA(X)(GETFILE(QUOTE EXPAND)(LIST X))))
(COLLECT(LAMBDA(X Y)(GETFILE(QUOTE COLLECT)(LIST X Y))))
```

```
(SOLVE(LAMBDA(X Y)(GETFILE(QUOTE SOLVE)(LIST X Y))))
(TRUNCATE(LAMBDA(X Y Z)(GETFILE(QUOTE TRUNCATE)(LIST X Y Z))))
(APARSE(LAMBDA(X)(GETFILE(QUOTE APARSE)(LIST X))))
(EVALUATE(LAMBDA(X Y)(GETFILE(QUOTE EVALUATE)(LIST X Y))))
(NORMPOLY(LAMBDA(X Y)(GETFILE(QUOTE NORMPOLY)(LIST X Y))))
(SPLIT(LAMBDA(X)(GETFILE(QUOTE SPLIT)(LIST X))))
(LIMIT(LAMBDA(X Y Z)(GETFILE(QUOTE LIMIT)(LIST X Y Z))))
))
DEFLIST((
(BRINGOVER(LAMBDA(X Y)(GETFILE(QUOTE BRINGOVER)X)))
(GROUP(LAMBDA(X Y)(GETFILE(QUOTE GROUP)X)))
(REPLACE(LAMBDA(X Y)(GETFILE(QUOTE REPLACE)X)))
(FACTOROUT(LAMBDA(X Y)(GETFILE(QUOTE FACTOROUT)X)))
)FEXPR)
DEFINE((
(REMPROPS(LAMBDA(LIST PROP)(MAPLIST LIST
(FUNCTION(LAMBDA(U)(REMPROP(CAR U)PROP))))))
(PFORMREAD(LAMBDA(E N)(APOFF(GETSUB(READF E(QUOTE PFORM))N))))
(PLS(LAMBDA(X Y)(EPLS X Y)))
(PRD(LAMBDA(X Y)(EPRD X Y)))
(NEG(LAMBDA(X Y)(ENEG X Y)))
(DVD(LAMBDA(X Y)(EDVD X Y)))
(EQN(LAMBDA(X Y)(EEQN X Y)))
(NILON(LAMBDA(X)(COND
((NULL X)(LIST NIL))
(T(CONS(CAR X)(NILON(CDR X)))))))
(DRV1(LAMBDA(X Y)(COND
((NULL(CDR X))Y)
(T(DRV1(CDDR X)(DRV2(CAR X)(CADR X)Y))))))
(DRV2(LAMBDA(X N Y)(COND
((ZEROP N)Y)
(T(DRV2 X(SUB1 N)(DIFF Y X))))))
(SUBSTITUTE(LAMBDA(X Y Z)
(SUBST1(SETTER(MKCLEANP Y))(SETTER(MKCLEANP Z))(MKCLEANP X))))
(SUBST1(LAMBDA(X Y Z)(COND
((NULL(CDR X))Z)
(T(SUBST1(CDR X)(CDR Y)(SUBST2(CAR X)(CAR Y)Z))) )))
(ALLSUMEXPAND(LAMBDA(X)(SUBSTA(QUOTE(SUMEXPAND EXP))
(QUOTE(SUM DOLLAR NIL))X)))
(COMMAND(LAMBDA(X)(PROG(U)
(COND((NULL X)(RETURN NIL)))
(EASSIGN(QUOTE LAST)(EVAL(APARSE X)NIL))
(RETURN NIL)
)))
))
DEFINE((
(DELETE(LAMBDA(X Y)(COND
((NULL X)NIL)
((EQUAL(CAR X)Y)(CDR X))
(T(CONS(CAR X)(DELETE(CDR X)Y))))))
(GETINSUB(LAMBDA(X N)(APOFF(GETSUB INPFORM N))))
))
DEFINE((
(EPRINT1(LAMBDA(X)(COND
((ATOM X)X)
((AND(MEMBER(CAR X)(QUOTE(PLS PRD PWR EQN)))(CDDR X))
(CONS(EPRINT1(CADR X))(EPRINT2(CAR X)
(MAPLIST(CDDR X)(FUNCTION(LAMBDA(U)(EPRINT1(CAR U))))))))
(T(MAPLIST X(FUNCTION(LAMBDA(U)(EPRINT1(CAR U)))))) ))
(EPRINT2(LAMBDA(X Y)(COND
((NULL(CDR Y))(CONS X Y))
(T(CONS X(CONS(CAR Y)(EPRINT2 X(CDR Y)))) )))
(SETTER(LAMBDA(X)(COND
((EQ(CAR X)(QUOTE ESET))(CDR X))
(T(LIST X NIL)) )))
(MKCLEANP(LAMBDA(X)(COND
((ATOM X)X)
((NULL(CDR X))(LIST NIL))
(T(CONS(MKCLEANP(CAR X))(MKCLEANP(CDR X)))) )))
))
DEFLIST((
(FORM(LAMBDA(X Y)(READF(CAR X)(QUOTE FORM))))
)FEXPR)
CSET(INPFORM NIL)
DEFINE((
(SUBST2(LAMBDA(X Y Z)(COND
((EQUAL Y Z)X)
((ATOM Z)Z)
((EQ(CAR Z)(QUOTE NAM))(CONS(CAR Z)
(SUBST3 X Y(CDR Z))))
((EQ(CAR Z)Y)(COND((ATOM X)(CONS X(MAPL
(CDR Z)(FUNCTION(LAMBDA(U)(SUBST2 X Y U))))))
((EQ(CAR X)(QUOTE NAM))(NTOLSUBST X
(CONS(NTOL X)(MAPL(CDR Z)(FUNCT*ON(LAMBDA(U)(SUBST2 X Y U)))) ))
(T(CONS(CAR Z)(MAPL(CDR Z)(FUNC.ION(LAMBDA(U)(SUBST2 X Y U))))))))
(T(CONS(CAR Z)(MAPL(CDR Z)(FUNCT'ION(LAMBDA(U)(SUBST2 X Y U))))) )))
))
DEFINE((
(SUBST3(LAMBDA(X Y Z)(COND
((NULL(CDDR Z))(CONS
(COND((ATOM(CAR Z))(SUBST2 X Y(CAR Z)))
(T(CONS(CAAR Z)(MAPL(CDAR Z)
(FUNCTION(LAMBDA(U)(SUBST2 X Y U)))))))
(CDR Z)))
(T(CONS(SUBST2 X Y(CAR Z))(SUBS'3 X Y(CDR Z)))))))
))
```

```
REMPROPS((NORMPOLY1 NORMPOLY2 NORMPOLY3 NORMPOLY4)EXPR)
DEFINE((
(NORMPOLY(LAMBDA(X Y)(GETFILE(QUOTE NORMPOLY)(LIST X Y))))
))
```

```
DEFINE((
(NORMPOLY(LAMBDA(Y XFREE)(SUBSTA
(QUOTE(NORMPOLY2 EXP(NORMPOLY1(CDDR EXP)(NORMPOLY4(CADR EXP)))))
(QUOTE(PLS DOLLAR NIL))Y)))
(NORMPOLY1(LAMBDA(X N)(COND
((NULL(CDR X))N)
(T(NORMPOLY1(CDR X)(MIN(NORMPOLY4(CAR X))N))) )))
(NORMPOLY4(LAMBDA(X)(COND
((EQUAL X XFREE)1)
((ATOM X)0)
((EQ(CAR X)(QUOTE PWR))(COND
((AND(EQUAL(CADR X)XFREE)(NUMBERP(CADDR X)))
(CADDR X))
(T 0)))
((EQ(CAR X)(QUOTE PRD))(NORMPOLY3(CDR X)))
(T 0) )))
(NORMPOLY3(LAMBDA(X)(COND
((NULL(CDR X))0)
(T(PLUS(NORMPOLY4(CAR X))(NORMPOLY3(CDR X)))) )))
(NORMPOLY2(LAMBDA(EXP N)(COND
((ZEROP N)EXP)
(T(SIMPLIFY(COPY(EPRD(EPWR XFREE N)(MULTIPLYTHROUGH EXP
(EPWR XFREE(MINUS N)))))) )))
))
```

```
CSET(SLASHCOMMA $$9/,9)
CSET(SCOPEHEIGHT 1000)
FLAG((TAU OMEGA INF PI THETA EP SUM ITG LP1 LP2 FTL RP1 RP2
LBR1 RBR1 LBR2 RBR2 LBK1 RBK1 LBK2 RBK2 SIGMA)ICHAR)
FLAG((SUM ITG LP1 LP2 RP1 RP2 DVD BAR)LCHAR)
DEFLIST((
(ONE 6)(TWO 12)
(SIGMA 8)(FTL 6)(TAU 6)(OMEGA 8)(INF 6)(PI 6)(THETA 6)(EP 4)
)W)
DEFLIST((
(ONE 4)(TWO 8)
(SIGMA 5)(FTL 4)(TAU 5)(OMEGA 5)(INF 5)(PI 5)(THETA 5)(EP 5)
)D)
DEFLIST((
(ONE 4)(TWO 8)
(SIGMA 3)(FTL 4)(TAU 3)(OMEGA 3)(INF 3)(PI 3)(THETA 4)(EP 3)
)H)
DEFLIST((
(ITG 6)
(SUM 11)
(LP1 4)
(LP2 5)
(LBK1 4)
(LBK2 5)
(LBR1 4)
(LBR2 5)
)W1)
DEFLIST((
(ITG 8)
)W2)
DEFLIST((
(ITG 14)
(SUM 7)
(LP1 6)
(LP2 12)
(LBK1 8)
(LBK2 12)
(LBR1 7)
(LBR2 12)
)L1)
DEFLIST((
(ONE 1)
(TWO 2)
)N)
DEFLIST((
(PAREN(6 LP1 13 LP2))
(BRACE(7 LBR1 13 LBR2))
(BRACKET(8 LBK1 13 LBK2))
)SIZELIST)
DEFLIST((
(CONCAT(LAMBDA(W H D Y)NIL))
(ABS(LAMBDA(W H D Y)(LIST 0 0 SIZE(LIST(QUOTE BAR)
0(PLUS H D)))(PLUS(TIMES 2(N SIZE))(W(CAR Y)))
0 SIZE(LIST(QUOTE BAR)0(PLUS H D)))))
(PWRUP(LAMBDA(W H D Y)NIL))
(SUPSUB(LAMBDA(W H D Y)NIL))
(TITLE(LAMBDA(W H D Y)NIL))
(LVCONCAT(LAMBDA(W H D Y)NIL))
(EVL(LAMBDA(W H D Y)(LIST(PLUS(TIMES 2(N SIZE))
(W(CADR Y)))
 0 SIZE(LIST(QUOTE BAR)0(PLUS H D)))))
(VCONCAT(LAMBDA(W H D Y)NIL))
(DVD(LAMBDA(W H D Y)(LIST (TIMES 1(N SIZE)) D SIZE(LIST(QUOTE DVD)
(PLUS W(TIMES -4(N SIZE)))))
))
(ITG(LAMBDA(W H D Y)(LIST U(PLUS D(MINUS
(TIMES(N SIZE)(GET(QUOTE ITG)(QUOTE L1))))))SIZE
(LIST(QUOTE ITG)(TIMES(N SIZE)(GET(QUOTE ITG)(QUOTE W1))))))))
(SUM(LAMBDA(W H D Y)(LIST(PLUS W(MINUS(W(CADDR Y)))
(TIMES -2(N SIZE)))(TIMES(N SIZE)(MINUS
(GET(QUOTE SUM)(QUOTE W1))))))
(PLUS D(MINUS(TIMES(N SIZE)(GET(QUOTE SUM)(QUOTE L1))))))
SIZE(LIST(QUOTE SUM)(TIMES(N SIZE)(GET(QUOTE SUM)(QUOTE W1)))))))
(PAREN(LAMBDA(W H D Y)(PROG(U V)
(SETQ U(GET(QUOTE PAREN)(QUOTE SIZER)))
(SETQ V(PLUS D(MINUS(TIMES(N SIZE)(GET U(QUOTE L1)))))))
(RETURN(LIST O V SIZE(LIST U(TIMES (N SIZE)(GET U(QUOTE W1))))
(PLUS W(MINUS(TIMES(N SIZE)(GET U(QUOTE W1)))))) V
SIZE(LIST(GET U(QUOTE MATE))(TIMES(N SIZE)(GET U(QUOTE W1))))))) ))
(PWR(LAMBDA(W H D Y)NIL))
(DELIMIT(LAMBDA(W H D Y)NIL))
(UNDELIMIT(LAMBDA(W H D Y)NIL))
)S)
DEFLIST((
(LP1 RP1)
(LP2 RP2)
(LPK1 RPK1)
(LPK2 RPK2)
(LPR1 RPR1)
(LPR2 RPR2)
)MATE)
DEFLIST((
(CONCAT(LAMBDA(W H D Y)(LIST O(PLUS D(MINUS(D(CAR Y))))
(W(CAR Y))(PLUS D(MINUS(D(CADR Y))))) ))
(ABS(LAMBDA(W H D Y)(LIST(TIMES 2(N SIZE))(PLUS D(MINUS(D(CAR Y))))))))
(PWRUP(LAMBDA(W H D Y)(LIST(CADDAR
(CDDDR(LAST(CADR Y))))
(CAR(CDDDAR(CDDDR(LAST(CADR Y))))))U O)))
(SUPSUB(LAMBDA(W H D Y)(LIST(PLUS W(MINUS
(SPACER(MAX(W(CAR Y))(W(CADR Y)))))))
(PLUS D(MINUS(D(CAR Y))))(MINUS(D(CAR(CDDDDR Y)))))
(PLUS W(MINUS(SPACER(MAX(W(CAR Y))(W(CADR Y)))))))
(PLUS D(H(CAR(CDDDDR Y))))
(PLUS(MAX(W(CADDR Y))(W(CADDDR Y)))
(MINUS(W(CADDR Y))))(PLUS D(H(CAR(CDDDDR Y))))
(PLUS(MAX(W(CADDR Y))(W(CADDDR Y)))(MINUS(W(CADDDR Y))))
(PLUS D(MINUS(D(CADDDR Y))))(MINUS(D(CAR(CDDDDR Y)))))
(SPACER(MAX(W(CADDR Y))(W(CADDDR Y))))
(MAX(D(CAR Y))(D(CADDDR Y))) ))
(LVCONCAT(LAMBDA(W H D Y)(LIST O O O (PLUS D(TIMES 2(N SIZE)))))))
(TITLE(LAMBDA (W H D Y)(LIST O(PLUS D(MINUS(D(CAR Y)))))
(HALF(PLUS W(W(CAR Y))(MINUS(W(CADR Y)))))(PLUS D(MINUS
(D(CADR Y))))))))
(EVL(LAMBDA(W H D Y)(LIST(PLUS W(MINUS(W (CAR Y))))O O O)))
(VCONCAT(LAMBDA(W H D Y)(LIST(HALF(PLUS W(MINUS
(W(CAR Y))))))(PLUS D(MINUS(D(CAR Y))))(HALF(PLUS W(MINUS(W(CADR Y))))))O)
))
(DVD(LAMBDA(W H D Y)(LIST(HALF(PLUS W(MINUS(W(CAR Y)))))
(PLUS(TIMES(N SIZE)2)D)(HALF(PLUS W(MINUS(W(CADR Y)))))O) ))
```

```
(ITG(LAMBDA(W H D Y)(LIST(PLUS W(MINUS(W(CAR Y))))
(PLUS D(MINUS(D(CAR Y))))
(TIMES(N SIZE)(GET(QUOTE ITG)(QUOTE W1)))
(PLUS D(MINUS(PLUS(H(CADR Y))(D(CADR Y)))(TIMES(N SIZE)
(GET(QUOTE ITG)(QUOTE L1)))))(TIMES(N SIZE)(GET(QUOTE ITG)(QUOTE W2)))
(PLUS(TIMES(N SIZE)(GET(QUOTE ITG)(QUOTE L1)))D)
(PLUS W(MINUS(SPACER(W(CADDDR Y)))(SPACER(MINUS(W(CAR Y))))))
(PLUS D(MINUS(D(CADDDR Y)))) )))
(SUM(LAMBDA(W H D Y)(LIST(HALF(PLUS W(MINUS(W(CAR Y)))
(TIMES -2(N SIZE))(MINUS
(W(CADDR Y))))))
(PLUS D(MINUS(TIMES(N SIZE)(GET(QUOTE SUM)(QUOTE L1)))-2(MINUS
(H(CAR Y)))
(MINUS(D(CAR Y)))))
(HALF(PLUS W(MINUS(W(CADR Y)))(TIMES -2(N SIZE))(MINUS(W(CADDR Y))))
(PLUS D(TIMES(N SIZE)(GET(QUOTE SUM)(QUOTE L1)))2)
(PLUS W(MINUS(W(CADDR Y))))
(PLUS D(MINUS(D(CADDR Y)))) ) ))
(PAREN(LAMBDA(W H D Y)(LIST(HALF(PLUS W(MINUS(W(CAR Y)))))
(PLUS D(MINUS(D(CAR Y))))) ))
(PWR(LAMBDA(W H D Y)(LIST 0 0(W(CAR Y))
(PLUS(H(CAR Y))(D(CAR Y))) ) ))
(DELIMIT(LAMBDA(W H D Y)(LIST 0 0) ))
(UNDELIMIT(LAMBDA(W H D Y)(LIST 0 0) ))
)A)
DEFLIST((
(CONCAT(LAMBDA(Y)(MAX(D(CAR Y))(D(CADR Y)))))
(ABS(LAMBDA(Y)(PLUS(TIMES 2(N SIZE))(D(CAR Y)))))
(PWRUP(LAMBDA(Y)(D(CADR Y))))
(SUPSUR(LAMBDA(Y)(PLUS(D(CAR(CDDDDR Y)))
(MAX(D(CAR Y))(D(CADDDR Y))))))
(TITLE(LAMBDA(Y)(MAX(D(CAR Y))(D(CADR Y)))))
(LVCONCAT(LAMBDA(Y)(PLUS(H(CAR Y))(D(CAR Y))(N SIZE))))
(EVL(LAMBDA(Y)(D(CADR Y))))
(VCONCAT(LAMBDA(Y)(PLUS(H(CADR Y))(D(CADR Y))(TIMES(N SIZE)20)(D(CAR Y))
)))
(OVD(LAMBDA(Y)(PLUS(N SIZE)(H(CADR Y))(D(CADR Y))) ))
(ITG(LAMBDA(Y)(MAX(PLUS(TIMES(N SIZE)(GET(QUOTE ITG)(QUOTE L1)))
(H(CADR Y))(D(CADR Y)))(D(CADDDR Y))) ))
(SUM(LAMBDA(Y)(MAX(PLUS 2(TIMES(N SIZE)(GET(QUOTE SUM)(QUOTE L1)))
(H(CADR Y))(D(CADR Y)))(D(CADDR Y))) ))
(PAREN(LAMBDA(Y)(MAX(D(CAR Y))(TIMES(N SIZE)(GET(GET(QUOTE PAREN)
(QUOTE SIZER))(QUOTE L1)))))))
(PWR(LAMBDA(Y)(D(CAR Y))))
(DELIMIT(LAMBDA(Y)(D(CAR Y)) ))
(UNDELIMIT(LAMBDA(Y)(D(CAR Y)) ))
)D)
DEFLIST((
(CONCAT(LAMBDA(Y)(PLUS(W(CAR Y))(W(CADR Y)))))
(ABS(LAMBDA(Y)(PLUS(TIMES 4(N SIZE))(W(CAR Y)))))
(PWRUP(LAMBDA(Y)(W(CADR Y))))
(SUPSUR(LAMBDA(Y)(PLUS(W(CAR(CDDDDR Y)))
(SPACER(MAX(W(CAR Y))(W(CADR Y))))(SPACER(MAX(W(CADDR Y))
(W(CADDDR Y))))))))
(TITLE(LAMBDA(Y)(MAX 1.0(PLUS(W(CAR Y))(W(CADR Y))))))
(LVCONCAT(LAMBDA(Y)(MAX(W(CAR Y))(W(CADR Y)))))
(EVL(LAMBDA(Y)(PLUS(TIMES(N SIZE)4)(W(CAR Y))(W(CADR Y)))))
(VCONCAT(LAMBDA(Y)(MAX(W(CAR Y))(W(CADR Y)))))
```

```
(DVD(LAMBDA(Y)(PLUS(TIMES 4(N SIZE))(MAX(W(CAR Y))(W(CADR Y))))))
(ITG(LAMBDA(Y)(PLUS(W(CADDDR Y))(TIMES(N SIZE)6)(W(CAR Y))(MAX
(PLUS(W(CADR Y))(TIMES(N SIZE)(GET(QUOTE ITG)(QUOTE W1))))
(PLUS(W(CADDR Y))(TIMES(N SIZE)(GET(QUOTE ITG)(QUOTE W2))))
))))
(SUM(LAMBDA(Y)(PLUS (TIMES 2(N SIZE)) (MAX(W(CAR Y))(W(CADR Y))
(TIMES(N SIZE)(GET(QUOTE SUM)
(QUOTE W1)))))(W(CADDR Y))) ))
(PAREN(LAMBDA(Y)(PROG (U)
(SETQ U(GET PARENKIND(QUOTE SIZELIST)))
(DEFLIST(LIST(LIST(QUOTE PAREN)(SIZER(MAX(H(CAR Y))
(D(CAR Y)))U)))(QUOTE SIZER))
(RETURN(PLUS(TIMES 2(N SIZE))(GET(GET(QUOTE PAREN)(QUOTE SIZER))(QUOTE W1
)))
(W(CAR Y)))) )))
(PWR(LAMBDA(Y)(PLUS(W(CAR Y))(W(CADR Y)))))
(DELIMIT(LAMBDA(Y)(W(CAR Y))))
(UNDELIMIT(LAMBDA(Y)(W(CAR Y))))
)W)
DEFLIST((
(CONCAT(LAMBDA(Y)(MAX(H(CAR Y))(H(CADR Y)))))
(ABS(LAMBDA(Y)(PLUS(TIMES 2(N SIZE))(H(CAR Y))))))
(PWRUP(LAMBDA(Y)(H(CADR Y))))
(SUPSUB(LAMBDA(Y)(PLUS(H(CAR(CDDDDR Y)))
(MAX(PLUS(H(CADR Y))(D(CADR Y)))
(PLUS(H(CADDR Y))(D(CADDR Y)))))))
(TITLE(LAMBDA(Y)(MAX(H(CAR Y))(H(CADR Y)))
))
(LVCONCAT(LAMBDA(Y)(PLUS(H(CADR Y))(D(CADR Y))(N SIZE))))
(EVL(LAMBDA(Y)(MAX(H(CADR Y))(PLUS(H(CAR Y))(D(CAR Y))(MINUS
(D(CADR Y)))))))
(VCONCAT(LAMBDA(Y)(H(CAR Y))))
(DVD(LAMBDA(Y)(PLUS(N SIZE)(H(CAR Y))(D(CAR Y)))))
(ITG(LAMBDA(Y)(MAX(PLUS(TIMES(N SIZE)(GET(QUOTE ITG)(QUOTE L1)))
(H(CADDR Y))(D(CADDR Y)))(H(CADDDR Y))) ))
(SUM(LAMBDA(Y)(MAX(PLUS 2 (TIMES(N SIZE)(GET(QUOTE SUM)(QUOTE L1)))
(H(CADR Y))(D(CADR Y)))(H(CADDR Y))) ))
(PAREN(LAMBDA(Y)(MAX(H(CAR Y))(TIMES (N SIZE)(GET(GET(QUOTE PAREN)(QUOTE
  SIZER))
(QUOTE L1)))))))
(PWR(LAMBDA(Y)(PLUS(H(CAR Y))(H(CADR Y))(D(CADR Y))) ))
(DELIMIT(LAMBDA(Y)(H(CAR Y)) ))
(UNDELIMIT(LAMBDA(Y)(H(CAR Y)) ))
)H)
CSET(ORDERLIST(PAREN NAM EQUAL FUNCTION FTL PWR FRT
 PRD EQUAL DVD ITG DRV NEG EQUAL PLS EQUAL
SUM))
FLAG((PLS PRD EQN DRV FRT NEG FTL NAM ESET)A)
FLAG((*SIN *COS *TAN *COT *LOG *SEC *CSC)TRANSCENDENTAL)
DEFLIST((
(ALIST(LAMBDA(X Y)Y))
)FEXPR)
DEFLIST((
(PAREN(LAMBDA(Y SIZE)(PARENO2(PAREND1 Y(QUOTE PAREN)))))
(TITLE(LAMBDA(Y SIZE)(DIMUP(LIST(QUOTE TITLE)(SCOPEWIDTHSET
(DIMDOWN(CAR Y)SIZE))(DIMDOWN(CADR Y)SIZE)NIL)SIZE)))
(ATOM(LAMBDA(X SIZE)(DIMUP1(LIST(QUOTE ATOM)(COND
((NUMBERP(CAR X))(DIMUP2(EXPLODE(CAR X))))
(T(CAR X)))(CADR X))
(COND((MEMBER(QUOTE ICHAR)(CAR X))(TIMES(N SIZE)(WG(CAR X))))(T
(TIMES(WG SIZE)
(ATOMD1(EXPLODE(CAR X))1))))
(COND((MEMBER(QUOTE ICHAR)(CAR X))(TIMES(N SIZE)(HG(CAR X))))(T(HG SIZE)
))
(COND((MEMBER(QUOTE ICHAR)(CAR X))(TIMES(N SIZE)(DG(CAR X))))(T(DG SIZE)
))SIZE)))
(EVL(LAMBDA(Y SIZE)(DIMUP(LIST(QUOTE EVL))(DIMDOWN(EVLD1 Y)
(QUOTE ONE))(DIMDOWN(NTOL Y)SIZE)NIL)SIZE)))
(FUNCTION(LAMBDA(X SIZE)(DIMDOWN(LIST(QUOTE DELIMIT)
(MKCONCAT(CAR X)
(COND((AND(MEMBER(QUOTE TRANSCENDENTAL)(CADAR X))
(OR(EQ(CAADR X))(QUOTE ATOM))(EQ(CAADR X)
(QUOTE DVD)))(AND(EQ(CAADR X)(QUOTE PRD))
(NULL(CDDDDR(CADR X)))))))
(MKCONCAT(MKSYM(QUOTE $$B/ B))(CADR X)))
(T(LIST(QUOTE PAREN)(NAMD2(CDR X))
(QUOTE (UNDELIMIT)))))))
(LAST X))SIZE)))
(PLS(LAMBDA(Y SIZE)(DELIMIT(LINER Y (DIMDOWN
(MKSYM(QUOTE /+))SIZE)SCOPEWIDTH))))
(PRD(LAMBDA(Y SIZE)(DELIMIT(LINER Y
(DIMDOWN(MKSYM(QUOTE *//))SIZE)
SCOPEWIDTH))))
(EQN(LAMBDA(Y SIZE)(DELIMIT(LINER Y
(DIMDOWN(MKSYM(QUOTE
/=))SIZE)SCOPEWIDTH))))
(SUM(LAMBDA(Y SIZE)(DIMUP(LIST(QUOTE SUM))(DIMDOWN(
MKCONCAT(CAR Y))
MKCONCAT(MKSYM(QUOTE /=))(CADR Y)))(QUOTE ONE))
(DIMDOWN(CADDR Y)(QUOTE ONE))
(DIMDOWN(CADDDR Y)
SIZE)(LAST Y))SIZE) ))
(ITG(LAMBDA(Y SIZE)(DIMUP(LIST(QUOTE ITG))(DIMDOWN(MKCONCAT
(MKSYM(QUOTE *D))
(CAR Y))SIZE)(DIMDOWN(CADR Y)(QUOTE ONE))(DIMDOWN(CADDR Y)
(QUOTE ONE))
(DIMDOWN(CADDDR Y)SIZE)(LAST Y))SIZE)))
(DRV(LAMBDA(Y SIZE)(COND
((EQUAL(CAR(NTOL Y))(QUOTE ATOM))
(DIMUP(LIST(QUOTE DVD)
(DIMDOWN(MKCONCAT(MKUNDELIMIT(DRVD1(MKSYM(QUOTE *D)))(DRVD2 Y)))
(NTOL Y))SIZE)
(DIMDOWN(DRVD3 Y)SIZE)(LAST Y))SIZE)))
(T(DIMDOWN(LIST(QUOTE DELIMIT)(MKCONCAT(MKUNDELIMIT
(LIST(QUOTE DVD)(MKUNDELIMIT(DRVD1(MKSYM(QUOTE *D)))(DRVD2 Y)))
(DRVD3 Y))(LAST Y)))
(NTOL Y))(LAST Y))SIZE)) )))
(NAM(LAMBDA(Y SIZE)
(NAMD4 Y(NAMD5 Y)NIL)))
(FRT(LAMBDA(Y SIZE)
(DIMUP(LIST(QUOTE DVD)(DIMDOWN(CAR Y)(QUOTE ONE))
(DIMDOWN(CADR Y)(QUOTE ONE))(LAST Y))SIZE)))
(FTL(LAMBDA(Y SIZE)(DIMDOWN(MKCONCAT(CAR Y)(MKSYM
(QUOTE FTL)))SIZE)))
(PWR(LAMBDA(Y SIZE)(COND
((AND(EQ(CAAR Y)(QUOTE NAM))(EQ(CAR(NTOL Y))(QUOTE ATOM))
(NOT(MEMBER(QUOTE NE)(NAMD5(CAR Y)))))
(DIMUP(MKPWRUP
(NAMD&(CDAR Y)(NAMD5(CAR Y))(LIST(CADR Y))))(SIZE))
(T(DIMUP(LIST(QUOTE PWR)(DIMDOWN(CAR Y)SIZE)
(DIMDOWN(CADR Y)(QUOTE ONE))(CADDR Y))SIZE)) )))
(NEG(LAMBDA(Y SIZE)(DELIM)((DIMDOWN(MKCONCAT(MKSYM
(QUOTE /-))(CAR Y))SIZE))))
(ESET(LAMBDA(Y SIZE)(DIMDOWN(LIST(QUOTE PAREN)(CONS
(QUOTE ESET1)Y)NIL)SIZE)))
(ESET1(LAMBDA(Y SIZE)(LINER Y(DIMDOWN
(MKSYM SLASHCOMMA )SIZE)(CSETQ SCOPEWIDTH
(PLUS SCOPEWIDTH -3 )))))
)DOWN)
DEFLIST((
(/*(NEG))
)ROMIT)
```

PICT2 LISP

```
DEFINE((
(EDPY(LAMBDA(Y )(PROG(PARENKIND U2 U1 W YPOS XPOS
X U V XNC YNC SXNC SYNC BXNC BYNC SIZE ATOM WIDTH SCOPEWIDTH)
(SETQ X(READF Y (QUOTE FORM)))
(SETQ YPOS 0)
(SETQ XPOS 512)
(SETQ SCOPEWIDTH 1023)
(SETQ U (PICTURE(LIST
(QUOTE TITLE)
(LIST(QUOTE PAREN)Y NIL)X NIL)))
(SETQ X NIL)
(PUNCH BLANK)
(FILEWRITE Y(QUOTE PFORM)U)
(DIMDOWN U(QUOTE TWO))
(COND((OR(GREATERP(W U)1023)
(GREATERP(PLUS(H U)(D U))1000))(GO A)))
(P JNCH DOLLAR)
(PUNCH(LIST(QUOTE MATHD)(LIST(QUOTE QUOTE)Y)))
(PUNCH(LIST(X U)(H U)(D U)))
(EDPY1(HALF(PLUS 1023(MINUS(W U))))
0
(PROG2(DIMUP5 U)U))
(PUNCH NIL)
B(PRIN1 DOLLAR)
(PRINT(QUOTE(ABSORBE)))
(RETURN NIL)
A(FILEDELETE Y(QUOTE P;ORM))
(PUNCH(QUOTE(EXPRESSION TOO LARGE)))
(GO B)
)))
(PICTURE(LAMBDA(X)(PAREN
      (PASS1 (MKCLEANP X)) NIL ORDERLIST T))) (W (LAMBDA (X) (COND ((NULL
 X) U) (T (CAR (LAST X)))))) (H (LAMBDA (X) (COND ((NULL X) 0) (T (CADR
(LAST X)))))) (D (LAMBDA (X) (C NO ((NULL X) 0) (T (CADDR (LAST X)))))) 
(A (LAMBDA (X) (GET X (QUOTE A);)) (S (LAMBDA (X) (GET X (QUOTE S)))) (W
G (LAMBDA (X) (GET X (QUOTE W)))) (HG (LAMBDA (X) (GET X (QUOTE H)))) (D
G (LAMBDA (X) (GET X (QUOTE D)))) (N (LAMBDA (X) (GET X (QUOTE N)))) (LA
ST (LAMBDA (X) (COND ((NULL (CDR X)) (CAR X)) (T (LAST (CDR X)))))) (MKS
YM (LAMBDA (X) (LIST (QUOTE ATOM) X (QUOTE (S))))) (MKCONCAT (LAMBDA (X
Y) (LIST (QUOTE CONCAT) X Y (QUOTE (UNDELIMIT))))) (PUTP (LAMBDA (EXP VA
LUE IND) (COND ((NULL (CDR EXP)) (LIST (CONS IND (CONS VALUE (CAR EXP)))
)) (T (CONS (CAR EXP) (PUTP (CDR EXP) VALUE IND)))))) (GETP (LAMBDA (EXP
 IND) (GET (LAST EXP) IND))) (MAPL (LAMBDA (X FN) (COND ((NULL (CDR X))
X) (T (CONS (FN (CAR X)) (MAPL (CDR X) FN)))))) (PASS1 (LAMBDA (Y) (COND
 ((AND (NUMBERP Y) (MINUSP Y)) (LIST (QUOTE NEG) (PASS1 (MINUS Y)) NIL))
((ATOM Y)(LIST(QUOTE ATOM)(COND(Y Y)(T(QUOTE UNDEFINED))) NIL))
((AND(EQ(CAR Y)(QUOTE FRT))
(MINUSP(CADDR Y)))(LIST(QUOTE NEG)(PASS1(LIST(CAR Y)(CADR Y)
(MINUS (CADDR Y))NIL))NIL))
((AND(OR(EQ(CAR Y)(QUOTE PLS))(EQ(CAR Y)(QUOTE PRD)))(NULL(CDDDR Y)))
(PASS1(CADR Y)))
((AND (EQ(CAR Y)(QUOTE PLS))(NUMBERP
      (CADR Y))) (PASS1 (CONS (CAR Y) (PASS13 (CADR Y) (CDDR Y))))) ((AN
D (EQ (CAR Y) (QUOTE PRD)) (EQU,L (CADR Y) -1) (NULL (CDDDDR Y))) (PASS1
 (LIST (QUOTE NEG) (CADDR Y) NIL))) ((EQ (CAR Y) (QUOTE PRD))
(COND((AND(NUMBERP(CADR Y))
(MINUSP(CADR Y)))
(LIST(QUOTE NEG)(PASS14
(COND((ONEP(MINUS(CADR Y)))(CDDR Y))
(T(CONS(MINUS(CADR Y))(CDDR Y))"I
(LIST NIL)(LIST NIL))NIL))
(T(PASS14 (CDR Y)(LIST NIL)(LIST NIL)))))
 ((AND (EQ (CAR Y) (QUOTE NAM)) (NOT (ATOM
(NTOL Y))) (NULL (A (CAR (NTOL Y)))))) (CONS (QUOTE FUNCTION) (MAPL (CONS
 (PASS15 Y) (CDR (NTOL Y))) (FUNCTION PASS1)))) ((NULL (A (CAR Y))) (CON
S (QUOTE FUNCTION) (MAPL Y (FUNCTION PASS1))))) ((AND (EQUAL (CAR Y) (QUO
TE DRV)) (ARGLIST Y)) (CONS (CAR Y) (MAPL (PASS11 (CDR Y) (ARGLIST Y)) (
FUNCTION PASS1)))).) (T (CONS (CAR Y) (MAPL (CDR Y) (FUNCTION PASS1)))))))
 (HIGHER (LAMBDA (OPABOVE OP LIST W) (COND ((EQ OP (QUOTE ABS)) NIL) ((A
ND (MEMBER OPABOVE (QUOTE (SUM ITG))) (NOT W)) NIL) ((AND (EQ OPABOVE (Q
UOTE PWR)) W) NIL) ((AND (MEMBER OP (QUOTE (SUM ITG))) W) NIL) ((EQ OP O
PABOVE) T) ((OR (MEMBER OPABOVE (QUOTE (FUNCTION NAM ABS PAREN))) (NULL
LIST)) NIL) ((EQUAL OPABOVE (CAR LIST)) (NOT (HIGHER1 OP (CDR LIST)))) (
(EQUAL OP (CAR LIST)) NIL) (T (HIGHER OPABOVE OP (CDR LIST) W))))) (HIGH
ER1 (LAMBDA (OP LIST) (COND ((EQUAL (CAR LIST) (QUOTE EQUAL)) (OR (EQUAL
 (CADR LIST) OP) (HIGHER1 OP (CDDR LIST)))) (T NIL))))) (PAREN (LAMBDA (Y
 X Z W) (COND ((EQUAL (CAR Y) (QUOTE ATOM)) Y) ((AND (EQ X (QUOTE PWR))
(EQ (CAR Y) (QUOTE FUNCTION)) (MEMBER (QUOTE TRANSCENDENTAL) (CADADR Y))
) (PAREN1 T (CONS (CAR Y) (PAREN2 (CDR Y) (CAR Y) Z)))) (T (PAREN1 (HIGH
ER X (CAR Y) Z W) (CONS (CAR Y) (PAREN2 (CDR Y) (CAR Y) Z))))))) (PAREN1
 (LAMBDA (X Y) (COND (X (LIST (QUOTE PAREN) Y NIL)) (T Y))))) (DIMDOWN (L
AMBDA (Y SIZE) (PROG (U) (COND ((NULL Y) (RETURN NIL))) (COND ((SETQ U (
GET (CAR Y) (QUOTE DOWN))) (RETURN (RPLACA (RPLACD Y (CDR (SETQ U (APPLY
 U (LIST (CDR Y) SIZE) (ALIST)))))) (CAR U))))) (RETURN (PROG2 (DIMDOWN1
(CDR Y) SIZE) (DIMUP Y SIZE)))).) (DIMUP1 (LAMBDA (X W H D SIZE) (COND (
(NULL (CDR X)) (LIST (CONS W (CONS H (CONS D (CONS W (CONS SIZE (CAR X))
)))))) (T (CONS (CAR X) (DIMUP1 (CDR X) W H D SIZE)))))) (DIMUP2 (LAMBDA
 (X) (PROG (U) (SETQ U X) (CLEARBUFF) A (COND ((NULL U) (RETURN (INTERN
(MKNAM))))) (PACK (QUOTE /)) (PACK (CAR U)) (SETQ U (CDR U)) (GO A)))) (
DIMUP3 (LAMBDA (W H D FNS FNA CDRX X) (DIMUP4 W H D (FNA W H D CDRX) (FN
S W H D CDRX) X))) (DIMUP4 (LAMBDA (W H D AL SL Y) (COND ((NULL (CDR Y))
 (RPLACA Y (CONS W (CONS H (CONS D (CONS AL (CONS SL (CAR Y)))))))) (T (
PROG2 (DIMUP5 (CAR Y)) (DIMUP4 W H D AL SL (CDR Y)))))))) (DIMUP (LAMBDA
(X SIZE) (PROG2 (DIMUP3 (APPLY (WG (CAR X)) (LIST (CDR X)) (ALIST)) (APP
LY (HG (CAR X)) (LIST (CDR X)) (ALIST)) (APPLY (DG (CAR X)) (LIST (CDR X
)) (ALIST)) (S (CAR X)) (A (CAR X)) (CDR X) X) X))) (HALF (LAMBDA (X) (Q
UOTIENT X 2)))))
DEFINE((
(NAMD5(LAMBDA(X)(GET(CADR(NTOL Y))(QUOTE NAM))))
))
DEFINE((
(ATOMD2(LAMBDA(X Y)(COND
((NULL X)NIL)
((EQ(CAR X)Y)(CDR X))
(T(CONS(CAR X)(ATOMD2(CDR X)Y))) )))
))
DEFINE((
(READF(LAMBDA(Y X)(PROG(U)
(COND((FILEGONE Y X)(ERROR(LIST
Y X(QUOTE UNDEFINED)(QUOTE FILE)))))
(FILESEEK Y X)
(SETQ U(READ))
(FILEENDRD Y X)
(RETURN U))))
))
```

PICT1 LISP

```
DEFINE
(((EDPY3 (LAMBDA(X )(COND
((LESSP X 3)(LIST(PLUS 131072 X)))
((ZEROP(QUOTIENT X 127)) (LIST 'PLUS 196608
X))) (T (CONS 65663 (EDPY3 (PLUS X -127)))))))) (PARENNEXT (LAMBDA (X) (S
ELECT X ((QUOTE PAREN) (QUOTE BRACE)) (QUOTE BRACKET)))) (PARENGREATER (
LAMBDA (X Y) (COND ((OR (EQ X (QUOTE BRACKET)) (EQ Y (QUOTE BRACKET))) (
QUOTE BRACKET)) (T (QUOTE BRACE))))) (PAREND2 (LAMBDA (Y) (PROG2 (SETQ P
ARENKIND (PARENGREATER (PARENNEXT (CDR Y)) PARENKIND)) (CAR Y)))) (PAREN
D1 (LAMBDA (Y PARENKIND) (CONS (DIMUP (LIST (QUOTE PAREN) (DIMDOWN (CAR
Y) SIZE) (CADR Y)) SIZE) PARENKIND))) (PASS17 (LAMBDA (X Y) (COND ((OR (
NULL (CDR Y))
(AND(NOT(EQ(CAR X)(QUOTE DEL)))(MEMBER(CAAR Y)(QUOTE(SUM DEL ITG))))))
(CONS X Y)) (T(CONS
(CAR Y) (PASS17 X (CDR Y))))))) (PASS16 (LAMBDA (X) (COND ((NULL (CDR X)
) X) ((MEMBER(CAAR X)(QUOTE(SUM DEL ITG)))(PASS17 (CAR X) (PASS16 (CDR X
)))) (T (CONS (CAR X) (PASS16 (CDR X)))))))) (PAREN2 (LAMBDA (EXP OPABOVE
 PLIST) (COND ((NULL (CDR EXP)) EXP) (T (CONS (PAREN (CAR EXP) OPABOVE P
LIST (NULL (CDDR EXP))) (PAREN2 (CDR EXP) OPABOVE PLIST))))))) (PASS15 (L
AMBDA (Y) (COND ((NULL (CDDR Y)) (CONS (CAAR Y) (CDR Y))) (T (CONS (CAR
Y) (PASS15 (CDR Y)))))))) (SPACER (LAMBDA (X) (COND ((ZEROP X) X) (T (PLU
S (N SIZE) X)))))) (DIMUP6 (LAMBDA (Y) (COND ((NULL (CDR Y)) (RPLACA Y (C
ONS 0 (CONS 0 (CONS 0 (CAR Y)))))) (T (DIMUP6 (CDR Y))))))) (MKPWRUP (LAM
BDA (Y) (LIST (QUOTE PWRUP) (PROG2 (DIMUP6 (CADDR Y)) (CADDR Y)) (PROG2
(RPLACA (CDDR Y) NIL) Y) NIL)))) (LINER1 (LAMBDA (X L CARX Y SCOPEWIDTH
) (PROG
 (U R V W Z) (SETQ U CARX) (SETQ V X) A (COND ((NULL (CDR V)) (RETURN U)
)) (SETQ R (CAAR V)) (SETQ W (DIMDOWN (CAR V) SIZE)) (SETQ Z (COND ((OR
(MEMBER L (GET (CADR Y) (QUOTE LOMIT))) (MEMBER R (GET (CADR Y) (QUOTE R
OMIT))))) NIL) (T Y))) (COND ((GREATERP (PLUS (W W) (W U) (W Z))
SCOPEWIDTH) (
RETURN (DIMUP (LIST (QUOTE VCONCAT) U (COND ((OR (GET (CADR Y) (QUOTE NO
MIT)) (NOT (MEMBER R (GET (CADR Y) (QUOTE ROMIT))))) (DIMUP (MKCONCAT (C
OPY Y) (LINER1 (CDR V) R W Y (PLUS SCOPEWIDTH (MINUS (W Y)))))) SIZE))
(T (LINER1
(CDR V)R W Y SCOPEWIDTH))) NIL) SIZE)))) (SETQ U (DIMUP (MKCONCAT U (CON
D ((NULL Z) W) (T (DIMUP (MKCONCAT (COPY Z) W) SIZE)))) SIZE)) (SETQ L R
) (SETQ V (CDR V)) (GO A)))) (ATOMD1 (LAMBDA (X Y) (COND ((NULL X) 0) ((
EQ (CAR X) (QUOTE *)) (ATOMD1 (CDR X) 0)) ((EQ (CAR X) (QUOTE =)) (ATOMD
1 (CDR X) 1)) ((AND(ZEROP Y)
(MEMBER (CAR X) (QUOTE (4 5 6 7)))) (ATOMD1 (CDR X) Y)) ((
EQ (CAR X) (QUOTE /)) (COND ((NUMBERP (CADR X)) (ATOMD1 (CDR X) Y)) (T (
ADD1 (ATOMD1(CDDR X)Y)))))
((AND(EQ(CAR X)DOLLAR)(EQ(CADR X)DOLLAR))
(ATOMD1(ATOMD2(CDDDR X)(CADDR X))Y))
                                        (T (ADD1 (ATOMD1 (CDR X) Y)))))) (NAMD6 (LA
MBDA (Y) (COND ((NULL Y) NIL) ((NULL (CDR Y)) (CAR Y)) (T (MKCONCAT (CAR
 Y) (MKCONCAT (MKSYM SLASHCOMMA) (NAMD6 (CDR Y)))))))) (NAMD7 (LAMBDA (Y
 X) (COND ((NULL (CDDR Y)) (CONS X (CDR Y))) (T (CONS (CAR Y) (NAMD7 (CD
R Y) X))))))) (NAMD4 (LAMBDA (Y X Z) (PROG (NE NW SW SE V U) (SETQ V Y) (
SETQ U X) A (COND ((NULL U) (RETURN (DIMUP (LIST (QUOTE SUPSUB) (DIMDOWN
 (NAMD1 (APPEND (REVERSE SE) V)) (QUOTE ONE)) (DIMDOWN (NAMD6 (APPEND Z
(REVERSE NE))) (QUOTE ONE)) (DIMDOWN (NAMD6 (REVERSE NW)) (QUOTE ONE)) (
DIMDOWN (NAMD6 (REVERSE SW)) (QUOTE ONE)) (DIMDOWN (NTOL Y) SIZE) NIL) S
IZE)))) (SELECT (CAR U) ((QUOTE NE) (SETQ NE (CONS (CAR V) NE))) ((QUOTE
 NW) (SETQ NW (CONS (CAR V) NW))) ((QUOTE SE) (SETQ SE (CONS (CAR V) SE)
)) ((QUOTE SW) (SETQ SW (CONS (CAR V) SW))) NIL) (SETQ V (CDR V)) (SETQ
U (CDR U)) (GO A)))) (PASS13 (LAMBDA (X Y) (COND ((NULL (CDR Y)) (CONS X
 Y)) (T (CONS (CAR Y) (PASS13 X (CDR Y))))))) (PASS14 (LAMBDA (X Y Z) (C
OND ((NULL (CDR X)) (COND ((NULL (CDR Z)) (CONS (QUOTE PRD) (MAPL (PASS1
6 Y) (FUNCTION PASS1)))) (T (LIST (QUOTE DVD) (COND ((NULL (CDR Y)) (QUO
TE (ATOM 1 NIL))) ((NULL (CDDR Y)) (PASS1 (CAR Y))) (T (CONS (QUOTE PRD)
 (MAPL (PASS16 Y) (FUNCTION PASS1))))) (COND ((NULL (CDDR Z)) (PASS1 (CA
R Z))) (T (CONS (QUOTE PRD) (MAPL (PASS16 Z) (FUNCTION PASS1))))) NIL)))
) ((AND (EQ (CAAR X) (QUOTE PWR)) (OR (AND (NUMBERP (CADDAR X)) (MINUSP
(CADDAR X))) (AND (EQ (CAR (CADDAR X)) (QUOTE PRD)) (EQUAL (CADR (CADDAR
 X)) -1) (NULL (CDDDDR (CADDAR X))))))) (PASS14 (CDR X) Y (PASS13 (COND (
(EQUAL (CADDAR X) -1) (CADAR X)) (T (LIST (QUOTE PWR) (CADAR X) (COND ((
NUMBERP (CADDAR X)) (MINUS (CADDAR X))) (T (CADDR (CADDAR X)))) NIL))) Z
))) (T (PASS14 (CDR X) (PASS13 (CAR X) Y) Z))))) (DELIMIT (LAMBDA (X) (P
ROG2 (DELIMIT1 X) X))) (DELIMIT1 (LAMBDA (X) (COND ((NULL (CDR X)) (RPLA
CA X (DELIMIT2 (CAR X)))) (T (DELIMIT1 (CDR X)))))) (DELIMIT2 (LAMBDA (X
) (COND ((NULL X) NIL) ((EQUAL (CAR X) (QUOTE UNDELIMIT)) (CDR X)) (T (C
ONS (CAR X) (DELIMIT2 (CDR X)))'))) (EDPY2 (LAMBDA NIL (PROG NIL (SETQ X
NC (PLUS XREL U1 (MINUS XPOS))) (SETQ YNC (PLUS YREL U2 (MINUS YPOS))) (
SETQ XPOS (PLUS XREL U1 WIDTH)) (SETQ YPOS (PLUS U2 YREL)) (COND ((NULL
(GET ATOM (QUOTE LCHAR))) (PROG2 (SETQ YNC (PLUS YNC SIZE)) (SETQ YPOS (
PLUS YPOS SIZE))))) (SETQ BXNC (TIMES 16 (QUOTIENT XNC 128))) (SETQ BYNC
 (TIMES 16 (QUOTIENT YNC 128))) (SETQ SXNC (REMAINDER XNC 128)) (SETQ SY
NC (REMAINDER YNC 128)) (SETQ V (COND ((EQ ATOM (QUOTE DVD)) (CONS 32848
 (EDPY3 (HALF WIDTH)))) ((EQ ATOM (QUOTE BAR)) (PROG2 (SETQ YPOS (PLUS Y
POS (CADDR (CADDDR W)))) (LIST 32848 (PLUS (TIMES (HALF (CADDR (CADDDR W
))) 256) 196608)))) (T (LIST (COND ((MEMBER (QUOTE ICHAR) ATOM) (COND ((
ZEROP SIZE) 49216) (T 49232))) (T (COND ((ZEROP SIZE) 24640) (T 24656)))
) ATOM)))) (COND ((AND (ZEROP SXNC) (ZEROP SYNC)) (GO D))) (SETQ V (CONS
 32832 (CONS (PLUS 131072 (TIMES 256 (ABS SYNC)) (SIGNY SYNC) (ABS SXNC)
 (SIGNX SXNC)) V))) D (COND ((AND (ZEROP BYNC) (ZEROP BXNC)) (GO E))) (S
ETQ V (CONS 32880 (CONS (PLUS 131072 (TIMES 256 (ABS BYNC)) (SIGNY BYNC)
 (ABS BXNC) (SIGNX BXNC)) V))) E        (PUNCH V)  (RETURN NIL)))) (DIMD
OWN1 (LAMBDA (Y SIZE) (COND ((NULL (CDR Y)) (CDR Y)) (T (PROG2 (COND ((N
ULL (CAR Y)) NIL) (T (DIMDOWN (CAR Y) SIZE))) (DIMDOWN1 (CDR Y) SIZE))))
)) (DIMUP5 (LAMBDA (X) (COND ((ATOM X) X) ((NULL (CDR X)) (RPLACA X (CDD
AR X))) (T (DIMUP5 (CDR X)))))) (MKCLEANP (LAMBDA (X) (COND ((ATOM X) X
) ((NULL (CDR X)) (LIST NIL)) (T (CONS (MKCLEANP (CAR X)) (MKCLEANP (CDR
 X))))))) (PASS11 (LAMBDA (Y X) (COND ((NULL (CDDR Y)) Y) (T (CONS (PASS
12 (CAR Y) X 1) (CONS (CADR Y) (PASS11 (CDDR Y) X)))))))) (PASS12 (LAMBDA
 (X Y N) (COND ((OR (ATOM X) (NULL (CDR Y))) X) ((EQUAL X (CAR Y)) N) (T
 (PASS12 X (CDR Y) (ADD1 N)))))) (SCOPEWIDTHSET (LAMBDA (Y) (PROG2 ( SET
Q SCOPEWIDTH (PLUS 1023 (MINUS (W Y)))) Y))) (ARGLIST (LAMBDA (Y) (COND
((ATOM (NTOL Y)) NIL) ((NULL (WG (CAR (NTOL Y)))) (CDR (NTOL Y))) ((AND
(EQUAL (CAR (NTOL Y)) (QUOTE NAM)) (NOT (ATOM (NTOL (NTOL Y)))) (NULL (W
G (CAR (NTOL (NTOL Y)))))) (CDR (NTOL (NTOL Y)))) (T NIL)))) (EVLD1 (LAM
BDA (Y) (COND ((NULL (CDDDDR Y)) (MKCONCAT (CAR Y) (MKCONCAT (MKSYM (QUO
TE /=)) (CADR Y)))) (T (LIST (QUOTE LVCONCAT) (MKCONCAT (CAR Y) (MKCONCA
T (MKSYM (QUOTE /=)) (CADR Y))) (EVLD1 (CDDR Y)) (QUOTE (UNDELIMIT))))))
) (NTOL (LAMBDA (X) (COND ((NULL (CDDR X)) (CAR X)) (T (NTOL (CDR X))))
) (MKUNDELIMIT (LAMBDA (Y) (PUTP Y (QUOTE UNDELIMIT) (QUOTE UNDELIMIT))))
) (LINER (LAMBDA (X Y WIDTH) (LINER1 (CDR X) (CAAR X) (DIMDOWN (CAR X) S
IZE) Y WIDTH))) (SIZER (LAMBDA (X Y) (COND ((OR (NULL (CDDR Y)) (NOT (LE
SSP (TIMES (N SIZE) (CAR Y)) X))) (CADR Y)) (T (SIZER X (CDDR Y)))))) (D
RVD1 (LAMBDA (BASE EXP) (COND ((AND (EQUAL (CAR EXP) (QUOTE ATOM)) (EQUA
L (CADR EXP) 1)) (LIST (QUOTE DELIMIT) BASE NIL)) (T (MKUNDELIMIT (LIST
(QUOTE PWR) BASE EXP (QUOTE (UNDELIMIT)))))))) (DRVD2 (LAMBDA (Y) (COND
((NULL (CDDR Y)) (LIST (QUOTE ATOM) 0 (QUOTE (UNDELIMIT)))) (T (EPLS (CA
DR Y) (DRVD2 (CDDR Y)))))) (DRVD3 (LAMBDA (Y) (COND ((NULL (CDDDDR Y))
(MKCONCAT (MKSYM (QUOTE *D)) (DRVD1 (CAR Y) (CADR Y)))) (T (MKCONCAT (MK
CONCAT (MKSYM (QUOTE *D)) (DRVD1 (CAR Y) (CADR Y))) (DRVD3 (CDDR Y))))))
) (NAMD1 (LAMBDA (Y) (COND ((NULL (CDDR Y)) NIL) ((NULL (CDDDR Y)) (CAR
Y)) (T (MKCONCAT (CAR Y) (MKCONCAT (MKSYM SLASHCOMMA) (NAMD1 (CDR Y))))
))) (NAMD2 (LAMBDA (Y) (COND ((NULL (CDDR Y)) (CAR Y)) (T (MKCONCAT (CAR
 Y) (MKCONCAT (MKSYM SLASHCOMMA) (NAMD2 (CDR Y)))))))) (EPLS (LAMBDA (X
Y) (COND ((AND (EQUAL (CAR X) (QUOTE ATOM)) (EQUAL (CAR Y) (QUOTE ATOM))
 (NUMBERP (CADR X)) (NUMBERP (CADR Y))) (LIST (CAR X) (PLUS (CADR X) (CA
DR Y)) (CADDR Y))) (T (LIST (QUOTE PLS) X Y (QUOTE (UNDELIMIT)))))))) (ED
PY1 (LAMBDA (XREL YREL X) (PROG (U UA) (COND ((NOT (OR (MEMBER (QUOTE S)
 (LAST X)) (MEMBER (QUOTE UNDELIMIT) (LAST X)))) (PUNCH        (QUOTE (1
96609)))) (COND ((EQUAL (CAR X) (QUOTE ATOM)) (GO B))) (COND ((EQUAL (
CAR X) (QUOTE UNDELIMIT)) (RPLACA (CDR X) (MKUNDELIMIT (CADR X))))) (SET
Q U (CDR X)) (SETQ UA (CAR (LAST X))) (COND ((SETQ W (CADR (LAST X))) (G
O C))) A (COND ((NULL (CDR U)) (GO D)) ((NULL (CAR U)) (GO E))) (EDPY1 (
PLUS XREL (CAR UA)) (PLUS YREL (CADR UA)) (CAR U)) E (SETQ U (CDR U)) (S
ETQ UA (CDDR UA)) (GO A) B (SETQ SIZE (SUB1 IN (CAR (CDADDR X))))) (SETQ
 ATOM (CADR X)) (SETQ WIDTH (CAADDR X)) (SETQ U1 0) (SETQ U2 0) (EDPY2)
(GO D) C (COND ((NULL W) (GO A))) (SETQ U1 (CAR W)) (SETQ U2 (CADR W)) (
SETQ SIZE (SUB1 IN (CADDR W)))) (SETQ ATOM (CAR (CADDDR W))) (SETQ WIDTH
 (CADR (CADDDR W))) (EDPY2) (SETQ W (CDDDDR W)) (GO C) D (COND ((NOT (OR
 (MEMBER (QUOTE S) (LAST X)) (MEMBER (QUOTE UNDELIMIT) (LAST X)))) (PUNC
H        (QUOTE (196610)) ))) (RETURN NIL)))) (SIGNX (LAMBDA (X) (COND (
(MINUSP X) 128) (T 0)))) (SIGNY (LAMBDA (X) (COND ((MINUSP X) 32768) (T
0))))) (ABS (LAMBDA (X) (COND ((MINUSP X) (MINUS X)) (T X)))))
))
```

```
                    PICOM2  LISP
COMPILE((ATOND2 NAND5 EOPY PICTURE W H D A S WG HG DG
N LAST MKSYM MKCONCAT PUTP GETP MAPL PASS1 HIGHER HIGHER1 PAREN
PAREN1 DIMDOWN DIMUP1 DIMUP2 DIMUP3 DIMUP4 DIMUP HALF))
```

PICOM1 LISP

```
DEFLIST((
(ALIST(LAMBDA(X Y)Y))
)FEXPR)
SPECIAL((SIZE V Y Z XNC YNC SXNC SYNC BYNC BXNC ATOM WIDTH XPOS YPOS
XREL YREL U1 U2 W))
COMMON((SCOPEWIDTH))
COMMON((FN PARENKIND DOLLAR BLANK FNS FNA ORDERLIST SLASHCOMMA))
COMPILE((EDPY3 PARENNEXT PARENGREATER PAREND2 PAREND1
PASS17 PASS16 PAREN2 PASS15 SPACER DIMUP6 MKPWRUP LINER1 ATOMD1
NAMD6 NAMD7 NAMD4 PASS13 PASS14 DELIMIT DELIMIT1 DELIMIT2 EDPY2
DIMDOWN1 DIMUP5 MKCLEANP PASS11 PASS12 SCOPEWIDTHSET ARGLIST
EVLD1 NTOL MKUNDELIMIT LINER SIZER DRVD1 DRVD2 DRVD3 NAMD1
NAMD2 EPLS EDPY1 SIGNX SIGNY ABS))
```

```
DEFLIST((
(REPLACE(LAMBDA(ARGS ALIST)(PROG(WFREE W)
(COND((CDDDR ARGS)(ERROR(QUOTE(LRONG NUMBER OF
ARGUMENTS FOR REPLACE)))))
(CSETQ INPNUM 0)
(CSETQ INPFORM(READF(EVAL(CAR ARGS)ALIST)(QUOTE PFORM)))
(SETQ WFREE(EVAL(SUBST(QUOTE GETINSUB)(QUOTE PFORMREAD)
(CADR ARGS))ALIST))
(COND((EQUAL(CADDR ARGS)(QUOTE(EQUOTE HOLE)))(GO A)))
(CSETQ INPNUM 0)
(CSETQ INPFORM(READF(EVAL(CAR ARGS)ALIST)(QUOTE PFORM)))
(EVAL(SUBST(QUOTE GETINSUB)(QUOTE PFORMREAD)(CADDR ARGS))ALIST)
A(SETQ W(APOFF(REPLACESUB INPFORM)))
(CSETQ INPFORM NIL)
(RETURN(SUBSTA(QUOTE(REPLACE1 EXP))
(QUOTE(DOLLAR(REPLACE2)))(SIMPLIFY W))) )))
)FEXPR)
DEFINE((
(REPLACE1(LAMBDA(X)(COND
((EQ(CAAR X)(QUOTE SUBMARK))
(COND(WFREE(CONS WFREE(REPLACE3(CDR X))))
(T(REPLACE3(CDR X)))))
(T(CONS(CAR X)(REPLACE1(CDR X))))) )))
(REPLACE3(LAMBDA(X)(COND
((NULL(CDR X))X)
((EQ(CAAR X)(QUOTE SUBMARK))(REPLACE3(CDR X)))
(T(CONS(CAR X)(REPLACE3(CDR X))))) )))
(REPLACE2(LAMBDA(X)(COND
((ATOM X)NIL)
((NULL(CDR X))NIL)
((EQ(CAAR X)(QUOTE SUBMARK))T)
(T(REPLACE2(CDR X)))) )))
))
```

```
                                        SETUP  LISP
FLAG((PLS PRD SUM TMS DVD FRT EWN STT
NEG PWR DRV ITG UNN INS NAM CND
GRT LSS GRE LSE XST LBD ANN NOT ORR
EVL ABS FTL INF SIN COS TAN COT SEC
CSC SINH COSH TANH COTH ASIN ACOS ATAN ACOT
NEXP NLOG)PERM)
CSET(RANDOMNUMBER 170598321)
DEFLIST(((SIN FSIN)(COS FCOS)(TAN FTAN)
(SEC FSEC)(CSC FCSC)
(COT FCOT)(SINH FSINH)(COSH FCOSH)(TANH FTANH)
(COTH FCOTH)(SECH FSECH)(CSCH FCSCH))FCODE)
DEFLIST(((E 552573617 3)(*E 836472034 4)
(PI 921361))CODE)
DEFLIST((
(SIN(COS *Y* NIL))
(COS (PRD -1 (SIN *Y* NIL)NIL))
(TAN(PWR(SEC *Y* NIL)2 NIL))
(COT(PRD -1(PWR(CSC *Y* NIL)2 NIL)NIL))
(SEC(PRD(TAN *Y* NIL)(SEC *Y* NIL)NIL))
(CSC(PRD -1(COT *Y* NIL)(CSC *Y* NIL)NIL))
(SINH(COSH *Y* NIL))
(COSH(SINH *Y* NIL))
(TANH (PWR(SECH *Y* NIL)2 NIL))
(COTH(PRD -1(PWR(CSCH *Y* NIL)2 NIL)NIL))
(SECH(PRD -1(SECH *Y* NIL)(TANH *Y* NIL)NIL))
(CSCH(PRD -1(CSCH *Y* NIL)(COTH *Y* NIL)NIL))
(ASIN(PWR(PLS 1(PRD -1(PWR *Y* 2 NIL)NIL)NIL)(PRD -1
(FRT 1 2 NIL)NIL)NIL))
(ACOS(PRD -1(PWR(PLS 1(PRD  -1(PWR *Y* 2 NIL)NIL)NIL)
(PRD -1(FRT 1 2 NIL)NIL)NIL)NIL))
(NLOG(PWR *Y* -1 NIL))
) DIFF)
```

SIMP1 LISP

```
DEFINE
(((SIMP34 (LAMBDA (X) (COND ((NULL (CDR X)) X) (T (CONS (CAAR X) (SIMP34
 (CDR X))))))) (SIMP31 (LAMBDA (X) (COND ((NULL (CDR X)) X) (T (CONS (CD
AR X) (SIMP31 (CDR X))))))) (SIMP33 (LAMBDA (X Y) (COND ((OR (EZEROP X)
(EONEP X)) (CONS X X)) (T (CONS X Y))))) (SIMP32 (LAMBDA (GARG Y) (GARG
Y))) (SIMPLIFY (LAMBDA (X) (CDR (SIMP2 X NIL 0))))
(SIMP2(LAMBDA(SX Y SIMPLEVEL)(
COND ((ATOM (CDR SX)) (COND ((EQUAL (CDR SX) SIMPLEVEL) (CAR SX)) (T (CO
NS (ECODE (CDAR SX) SIMPLEVEL) (CDAR SX))))) ((OR (EENUMBERP SX) (ATOM S
X)) (SIMP3 NIL SX Y)) ((EQUAL (CAR SX) (QUOTE PWR)) (SIMP6 (SIMP3 (CAR S
X) (LIST (SIMP2 (CADR SX) (CAR SX) SIMPLEVEL) (SIMP2 (CADDR SX) NIL (LOG
XOR 1 SIMPLEVEL))) Y) SX)) (T (SIMP6 (SIMP3 (CAR SX) (MAPL (CDR SX) (FUN
CTION (LAMBDA (U) (SIMP2 U (CAR SX) SIMPLEVEL)))) Y) SX))))) (SIMP3 (LAM
BDA (X Y Z) (COND ((NOT (NULL (GET X (QUOTE FCODE)))) (SIMP33 (SIMP32 (G
ET X (QUOTE FCODE)) (CAAR Y)) (CONS X (CONS (CDAR Y) (CDR Y))))) ((NOT (
NULL (GET X (QUOTE NSIMP)))) (SIMP33 (SIMP32 (GET X (QUOTE NCODE)) (SIMP
34 Y)) (SIMP32 (GET X (QUOTE NSIMP)) (SIMP31 Y)))) (T (SELECT X
((QUOTE ITG)(SIMP8 X Y))
((QUOTE DRV)(SIMP8 X Y))
((QUOTE
PLS) (SIMP5 Y Z (CONS 0 0) NIL (FUNCTION DPLS) (FUNCTION DPRD) X)) ((QUO
TE PRD) (SIMP5 Y Z (CONS 1 1) NIL (FUNCTION DPRD) (FUNCTION DPWR) X)) ((
QUOTE PWR) (SIMP7 Y Z)) ((QUOTE NIL) (CONS (GETC Y) Y)) (CONS (FGFN (GET
C X) Y) (CONS X (SIMP31 Y)))))))) (SIMP5 (LAMBDA (X Y SU V FNW FNZ R) (C
OND ((NULL (CDR X))(PROG2(SETQ  (SIMP56 V))
 (COND ((NULL V) SU) ((OR (EQUAL Y R) (AND (EQUAL R (
QUOTE PRD)) (EQUAL Y (QUOTE PWR)))) (CONS V SU)) (T (PROG (S) (SETQ S (S
IMP82 (SIMP53 V (COND ((EQUAL R (QUOTE PLS)) (FUNCTION APLS)) (T (FUNCTI
ON APRD))) FNZ) (COND ((EQUAL R (QUOTE PLS)) (FUNCTION MKPLS)) (T (FUNCT
ION MKPRD)))))) (COND ((AND (NOT (ONEP (CAR SU)))
(EQUAL Y(QUOTE PLS)) (
EQUAL (CADR S) (QUOTE PLS)) (EQUAL R (QUOTE PRD))) (RETURN (SIMP54 (CONS
 NIL (CONS NIL (MAPL (CDDR S) (FUNCTION (LAMBDA (H) (EPRD (CDR SU) H))))
)) (CONS 0 0) NIL)))) (RETURN (FNW SU S)))))))((AND (ATOM (CAAR X)) ( EN
UMBERP (CDAR X))) (SIMP5 (CDR X) Y (FNW SU (CAR X)) V FNW FNZ R)) ((EENU
MBERP (CDDAR X)) (SIMP5 (CDR X) Y (FNW SU (CDAR X)) (SIMP51 (CAAR X) V
 (LOGXOR SIMPLEVEL(COND((EQ R(QUOTE PRD))1)(T 0))))
FNW FNZ R)) ((AND (EQUAL R (QUOTE PLS)) (EQUAL (CADAR X) R)) (SIMP5 (CON
S (SIMP54 (CAR X) (CONS 0 0) NIL) (CDR X)) Y SU V FNW FNZ R)) ((AND (EQU
AL R (QUOTE PRD)) (EQUAL (CADAR X) R)) (SIMP5 (CONS (SIMP54 (CAR X) (CON
S 1 1) T) (CDR X)) Y SU V FNW FNZ R)) ((AND (EQUAL R (QUOTE PRD)) (EQUAL
 (CADAR X) (QUOTE PWR))) (SIMP5 (CONS (SIMP55 (CAR X) 1) (CDR X)) Y SU V
 FNW FNZ R)) ((AND (EQUAL R (QUOTE PLS)) (EQUAL (CADAR X) (QUOTE PRD)) (
EENUMBERP (CADDAR X))) (SIMP5 (CONS (SIMP55 (LIST NIL NIL (MKPRD (CDDDAR
 X)) (CADDAR X)) 0) (CDR X)) Y SU V FNW FNZ R)) (T (SIMP5 (CDR X) Y SU (
SIMP52 (COND ((ATOM (CAAR X)) (LIST (CAR X) (CONS 1 1))) (T (CAAR X))) V
 (LOGXOR SIMPLEVEL(COND((EQ R(QUOTE PRD))1)(T 0)))
) FNW FNZ R)))))
(SIMP51(LAMBDA(X Y SIMPLEVEL)(COND((NULL X)Y)(T(SIMP51
(CDR X)(SIMP52(CAR X)Y SIMPLEVEL)SIMPLEVEL))))))
(SIMP52 (LAMBDA (X Y SIMPLEVEL)
(COND((NULL Y)(LIST
 X)) ((SIMPEQUAL(CAAR X) (CAAAR Y))(CONS (LIST (CAAR Y)(DPLS
(COND((MINUSP(SIMPEQUAL(CAAR X)(CAAAR  Y)))
(SIMPNEG(CADR X))) (T(CADR X)))
(CADAR Y)))
        (CDR Y))) (T (CONS (CAR Y) (SIMP52 X (CDR Y)SIMPLEVEL)))))
))
```

```
                          SIMP2  LISP
DEFINE((
(SIMP53 (LAMBDA(X FNWW FNZ)(PROG(U V)
(COND((NULL X)(RETURN(LIST NIL))))
(SETQ U(FNZ(CAAR X)(CADAR X)))
(SETQ V(SIMP53(CDR X) FNWW FNZ))
(COND((EENUMBERP(CDR U))(GO A)))
(COND((NULL(CDR V))(RETURN(LIST(CAR U)(CDR U)NIL))))
(RETURN(CONS(FNWW(CAR U)(CAR V))(CONS(CDR U)(CDR V))))
A(SETQ SU(FNW U SU))
(RETURN V)
)))
 (SIMP54 (LAMBDA (X Y Z) (PROG (U V W) (SETQ U Y) (SETQ V (CDDR X)) A
(COND ((NULL (CDR V)) (RETURN (CONS W U))) ((EENUMBERP (CAR V)) (SETQ U
(CONS (GETC (CAR V)) (CAR V)))) ((AND (NULL Z) (EQUAL (CAAR V) (QUOTE PR
D)) (EENUMBERP (CADAR V))) (SETQ W (CONS (LIST (CONS (ECODE (MKPRD (CDDA
R V)) SIMPLEVEL) (MKPRD (CDDAR V))) (CONS (GETC (CADAR V)) (CADAR V))) W
))) ((AND (NOT (NULL Z)) (EQUAL (CAAR V) (QUOTE PWR))) (SETQ W (CONS (LI
ST (CONS (ECODE (CADAR V) SIMPLLVEL) (CADAR V)) (CONS (ECODE (CADDAR V)
(LOGXOR 1
SIMPLEVEL))(CADDAR V))) W))) (T (SETQ W (CONS (LIST (CONS (ECODE (CAR V)
 SIMPLEVEL) (CAR V)) (CONS 1 1)) W)))) (SETQ V (CDR V)) (GO A)))) (SIMP5
5 (LAMBDA (X Y) (CONS (LIST (LIST (CONS (ECODE (CADDR X) SIMPLEVEL) (CAD
DR X)) (CONS (ECODE (CADDDR X) (COND ((ZEROP Y) SIMPLEVEL) (T (LOGXOR SI
MPLEVEL 1)))) (CADDDR X)))) (CONS Y Y)))) (SIMP7 (LAMBDA (X Y) (PROG (U
V) (COND ((ATOM (CAAR X)) (GO A))) (SETQ V (SIMP71 (CAAR X)
(CADR X)(LOGXOR SIMPLEVEL 1))) (
SETQ U (DPWR (CDAR X) (CADR X))) (COND ((OR (EQUAL Y (QUOTE PWR)) (EQUAL
 Y (QUOTE PRD))) (RETURN (COND ((EENUMBERP (CDR U)) (CONS V U))
((EQ(CADR U)(QUOTE PWR))(CONS(COND((EQ(CAADDR U)(QUOTE FRT))
(CONS(SIMP57(CADADR(CDR U))(CADDDR U))
(CONS(SIMP57(CADDAR(CDDR U))(ENEG(CADDDR U)))V)))
(T(CONS(SIMP57(CADDR U)(CADDDR U))V)))(CONS 1 1)))
(T(CONS
 (CONS (LIST U (CONS 1 1)) V) (CONS 1 1))))))) (RETURN (DPRD (SIMP82 (SI
MP53 V (FUNCTION APRD) (FUNCTION DPWR)) (FUNCTION MKPRD)) U)) A (COND ((
EQUAL (CADAR X) (QUOTE PWR)) (RETURN (SIMP7 (CONS (SIMP55 (CAR X) 1) (CD
R X)) Y))) (COND ((EQUAL (CADAR X) (QUOTE PRD)) (RETURN (SIMP7 (CONS (S
IMP54 (CAR X) (CONS 1 1) T) (CDR X)) Y)))) (RETURN (DPWR (CAR X) (CADR X
)))))) (SIMP71 (LAMBDA (X Y SIMPLEVEL) (COND ((NULL X) NIL) (T (CONS
(LIST (CAAR X
) (SIMP5 (LIST (CADAR X) Y NIL) NIL (CONS 1 1) NIL (FUNCTION DPRD) (FUNC
TION DPWR) (QUOTE PRD))) (SIMP71 (CDR X) Y SIMPLEVEL))))))
(SIMP6(LAMBDA(X Y)(CAR
   (RPLACD (RPLACA Y X) SIMPLEVEL)))) (SIMP82 (LAMBDA (X FNW) (CONS (CAR
 X) (FNW (CDR X))))))
DEFINE((
(SIMP56(LAMBDA(X)(COND
((NULL X)NIL)
((OR(EZEROP(CDADAR X))(EZEROP(CAADAR X)))(SIMP56(CDR X)))
(T(CONS(CAR X)(SIMP56(CDR X)))) )))
))
DEFINE((
(SIMP57(LAMBDA(X Y)(LIST(CONS(ECODE X SIMPLEVEL)X)
(CONS(ECODE Y(LOGXOR SIMPLEVEL 1))Y))))
))
DEFINE((
(SIMP8(LAMBDA(X Y)(COND
((OR(EMINUSP(CDR(NTOL Y)))(AND(EQ(CADR(NTOL Y))(QUOTE PRD))
(EMINUSP(CADDR(NTOL Y)))))
(SIMPNEG(CONS(FGFN(GETC X)Y)(CONS X(SIMP31(NTOLSUBST
  Y(SIMPNEG(NTOL Y))))))))
(T(CONS(FGFN(GETC X)Y))(CONS X(SIMP31 Y)))) )))
(SIMPNEG(LAMBDA(X)(DPRO(CONS(ECODE =1 SIMPLEVEL)-1)X)))
(SIMPEQUAL(LAMBDA(X Y))COND
((EQUAL X Y)1)
((EQUAL(ANEG X)Y)-1)
(T NIL))))
))
```

```
COMMON((FNW FNWW FNZ AFUN EEFUN FN NUMFUN NUMP GARG))
SPECIAL((SU SIMPLEVEL SX))
COMPILE((SIMP31 SIMP33 SIMP32 S.MPLIFY SIMP2
SIMP3 SIMP5 SIMP51 SIMP52
SIMP34))
```

```
                              SIMP2   COMP
COMPILE((SIMP53 SIMP54 SIMP55 SIMPT SIMPT1 SIMP6 SIMP02
SIMP56 SIMP57 SIMP8 SIMPEQUAL SIMPNEG))
```

```
DEFINE
(((SOLVE21 (LAMBDA (X Y Z) (COND ((EQUAL (CAR Z) X) (CADR Z)) (T (LIST Y
 Z NIL)))))
(SOLVE (LAMBDA (Y X) (COND ((NOT (EQUAL
   (CAR Y) (QUOTE EQN))) (ERROR (CONS Y (QUOTE (IS NOT AN EQUATION)))))
(T(SIMPLIFY(SUBSTITUTE(SOLVE1(SUBSTITUTE
(CADR Y)(QUOTE SOLVE )X)(SUBSTITUTE(CADDR Y)(QUOTE SOLVE)X)
(QUOTE SOLVE1)X(QUOTE SOLVE)) )) )))
(SOLVE1 (LAMBDA
 (Y Z X) (PROG (U V) (SETQ U (SOLVE3 (MKP (CDR (SIMP2(EXPAND Y) NIL 0)))
 X)) (SETQ V (SOLVE3 (MKP (CDR (SIMP2(EXPAND Z) NIL 0))) X)) (RETURN (SO
LVE2 (EPLS (CAR U) (ENEG (CAR V))) (EPLS (ENEG (CDR U)) (CDR V)) X)))))
(SOLVE2 (LAMBDA (Y Z X) (COND ((EQUAL Y X) (LIST (QUOTE EQN) Y Z NIL)) (
T (SELECT (CAR Y) ((QUOTE PLS) (PROG (U) (SETQ U (SOLVE5 (CDR Y) (SOLVE5
1 (CADR Y) X) (LIST NIL) X)) (COND ((NULL U) (RETURN (LIST (QUOTE EQN) Y
 Z NIL)))) (RETURN (SOLVE2 (CAR U) (EDVD Z (CDR U)) X)))) ((QUOTE PRD) (
PROG (U) (SETQ U (SOLVE4 (CDR Y) (LIST NIL) NIL X)) (COND ((NULL U) (RET
URN (LIST (QUOTE EQN) Y Z NIL)))) (RETURN (SOLVE2 (CAR U) (EDVD Z (CDR U
)) X)))) ((QUOTE PWR) (COND ((NOT (DEPEND (CADDR Y) X)) (SOLVE6 (CADR Y)
 (EPWR Z (EDVD 1 (CADDR Y))) X)) (T (SOLVE6 (ENLOG Y) (ENLOG Z) X)))) ((
QUOTE NLOG) (SOLVE6 (CADR Y) (EPWR (QUOTE E) Z) X)) ((QUOTE SIN) (SOLVE6
 (CADR Y) (SOLVE21 (QUOTE SIN) (QUOTE ASIN) Z) X)) ((QUOTE COS) (SOLVE6
(CADR Y) (SOLVE21 (QUOTE COS) (QUOTE ACOS) Z) X)) ((QUOTE TAN) (SOLVE6 (
CADR Y) (SOLVE21 (QUOTE TAN) (QUOTE ATAN) Z) X)) ((QUOTE ASIN) (SOLVE6 (
CADR Y) (SOLVE21 (QUOTE ASIN) (QUOTE SIN) Z) X)) ((QUOTE ACOS) (SOLVE6 (
CADR Y) (SOLVE21 (QUOTE ACOS) (QUOTE COS) Z) X)) ((QUOTE ATAN) (SOLVE6 (
CADR Y) (SOLVE21 (QUOTE ATAN) (QUOTE TAN) Z) X)) (LIST (QUOTE EQN) Y Z N
IL)))))) (SOLVE3 (LAMBDA (Y X) (COND ((EQUAL (CAR Y) (QUOTE PLS)) (SOLVE
31 (CDR Y) (LIST NIL) (LIST NIL) X)) ((DEPEND Y X) (CONS Y 0)) (T (CONS
0 Y))))) (SOLVE31 (LAMBDA (Y U V X) (COND ((NULL (CDR Y)) (CONS (MKPLS U
) (MKPLS V))) ((DEPEND (CAR Y) X) (SOLVE31 (CDR Y) (CONS (CAR Y) U) V X)
) (T (SOLVE31 (CDR Y) U (CONS (CAR Y) V) X))))) (SOLVE4 (LAMBDA (Y U V X
) (COND ((NULL (CDR Y)) (CONS V (MKPRD U))) ((DEPEND (CAR Y) X) (COND ((
NULL V) (SOLVE4 (CDR Y) U (CAR Y) X)) (T NIL))) (T (SOLVE4 (CDR Y) (CONS
 (CAR Y) U) V X)))))) (SOLVE5 (LAMBDA (Y R U X) (COND ((NULL (CDR Y)) (CO
NS R (MKPLS U))) (T (PROG (V) (SETQ V (SOLVE52 (CAR Y) R X)) (COND ((NUL
L V) (RETURN NIL))) (RETURN (SOLVE5 (CDR Y) R (CONS V U) X)))))))) (SOLVE
51 (LAMBDA (Y X) (COND ((EQUAL (CAR Y) (QUOTE PRD)) (SOLVE511 (CDR Y) X)
) (T Y)))) (SOLVE511 (LAMBDA (Y X) (COND ((DEPEND (CAR Y) X) (CAR Y)) (T
 (SOLVE511 (CDR Y) X))))) (SOLVE52 (LAMBDA (Y R X) (COND ((EQUAL Y R) 1)
 ((NOT (EQUAL (CAR Y) (QUOTE PRD))) NIL) (T (SOLVE521 (CDR Y) NIL (LIST
NIL) R X)))))) (SOLVE521 (LAMBDA (Y B U R X) (COND ((NULL (CDR Y)) (COND
((NULL B) NIL) (T (MKPRD U)))) ((EQUAL (CAR Y) R) (SOLVE521 (CDR Y) T U
R X)) ((DEPEND (CAR Y) X) NIL) (T (SOLVE521 (CDR Y) B (CONS (CAR Y) U) R
 X)))))) (SOLVE6 (LAMBDA (Y Z X) (PROG (U) (SETQ U (MKP (CDR (SIMP2 (MULT
2 Y) NIL 0)))) (SETQ U (SOLVE3 U X)) (RETURN (SOLVE2 (CAR U) (EPLS (ENEG
 (CDR U)) Z) X)))))))
```

```
DEFINE((
(SPLIT(LAMBDA(X)(MKCLEANP(SPLIT2(SPLIT1 X)100))))
(SPLIT1(LAMBDA(X)(COND
((ATOM X)X)
(T(SPLIT3(CONS(CAR X)(MAPL(CDR X)(FUNCTION SPLIT1)))0)) )))
(SPLIT3(LAMBDA(X N)(COND
((NULL(CDR X))(LIST N))
(T(CONS(CAR X)(SPLIT3(CDR X)(PLUS(SPLIT4(CAR X))N)))) )))
(SPLIT2(LAMBDA(X N)(COND
((ATOM X)X)
((ZEROP(LAST X))(EASS(NEWNAME)(MKCLEANP X)))
((NOT(GREATERP (LAST X)N))X)
(T(PROG(U M R)
(SETQ U X)
(SETQ M(LAST U))
A(SETQ R(SPLIT6 U))
(COND((LESSP R 2)(RETURN
(EASS(NEWNAME)(MKCLEANP X)))))
(SETQ M(DIFFERENCE M R))
(COND((LESSP M N)(RETURN(SPLIT7 U R(DIFFERENCE N M)))))
(SETQ U(SPLIT5 U R))
(GO A))) )))
(SPLIT4(LAMBDA(X)(COND
((ATOM X)1)
(T(LAST X)) )))
(SPLIT6(LAMBDA(X)(COND
((NULL(CDDR X))(SPLIT4(CAR X)))
(T(MAX(SPLIT4(CAR X))(SPLIT6(CDR X)))) )))
(SPLIT7(LAMBDA(U R M)(COND
((NULL (CDR U))U)
((EQUAL(SPLIT4(CAR U))R)(CONS(SPLIT2(CAR U)M)
(SPLIT7(CDR U)-1 M)))
(T(CONS(SPLIT2(CAR U)(LAST(CAR U)))(SPLIT7(CDR U)R M))) )))
(SPLIT5(LAMBDA(U R)(COND
((NULL(CDR U))U)
((EQUAL R(SPLIT4(CAR U)))(CONS(SPLIT8(CAR U))(CDR U)))
(T(CONS(CAR U)(SPLIT5(CDR U)R))) )))
(SPLIT8(LAMBDA(U)(COND
((NULL(CDR U))(LIST 0))
(T(CONS(CAR U)(SPLIT8(CDR U)))) )))
))
```

```
DEFINE((
(SUMFACH(LAMBDA(XFREE)(COND
((AND(EQ(CAR XFREE)(QUOTE SUM))
(EQ(CAR(NTOL XFREE))(QUOTE PLS)))
(CONS(QUOTE PLS)(MAPL(CDR(NTOL XFREE))
(FUNCTION(LAMBDA(U)(NTOLSUBST XFREE U)))))))
(T XFREE) )))
))
```

```
DEFINE((
(SUMEXPAND(LAMBDA(X)(COND
((AND(EQ(CAR X)(QUOTE SUM))(NUMBERP(CADDR X))(NUMBERP(CADDDR X)))
(MKPLS(SUMEXPAND1(CADR X)(CADDR X)
(CADDDR X)(CADDDR(CDR X)))))
(T X))))
(SUMEXPAND1(LAMBDA(I N1 N2 X)(COND
((GREATERP N1 N2)(QUOTE(NIL)))
((EQUAL N1 N2)(LIST(SUBST N1 I X)NIL))
(T(CONS(SUBST N1 I X)(SUMEXPAND1 I(ADD1 N1)
N2 X))))))
))
```

RUNCA LISP

```
DEFINE
(((TRNK111 (LAMBDA (Y X N) (TRNK21 (TRNK8 (TRNK7(MKP Y)X) X) X N))) (  T
RNK122 (LAMBDA (  U N X) (PROG NIL (FILEWRITE (QUOTE LISPT) (QUOTE LISP)
 (QUOTE TRNK123))(FILEAPND (QUOTE LISPT) (QUOTE LISP) (LIST U N X))
(FILEAPND(QUOTE LISPT)(QUOTE LISP)(QUOTE CHAIN))
(FILEAPND(QUOTE LISPT)(QUOTE LISP)(QUOTE((RESUME GOO3)))))
(CHAIN (QUOTE ((SAVE GOO3
 T)(RESUME LISPT LISPT))))
(FILESEEK(QUOTE LISPTR)(QUOTE LISP))
(SETQ U(READ)) (FILEDELETE (QUOTE LISPT) (QUOTE LISP))
 (RETURN U)))) (TRUNCATE(LAMBDA (Y X N) (PROG (U) (DEFLIST (LIST (LIST (Q
UOTE MIN) (QUOTE EMIN))) (QUOTE NCODE)) (DEFLIST (LIST (LIST (QUOTE MIN)
 (QUOTE (LAMBDA (Y) (PROG (U) (SETQ U (CONS (QUOTE MIN) Y)) (COND ((NULL
 (LAST (MKP U))) (RETURN (EMIN (CDR U)))) (RETURN U)))))) (QUOTE NSIMP)
)(RETURN(SIMPLIFY(TRNK122(TRNK11 Y X N)N X))) )))
(TRNKTEST(LAMBDA(Y N X)(DEFLIST(LIST
 (LIST Y (TRNK11 (GET Y (QUOTE FORM)) X N))) (QUOTE FORM)))) (TRNK85 (LA
MBDA (Y X) (COND ((EZEROP (CADDR Y)) 1) ((AND (NUMBERP (CADDR Y)) (FIXP
(CADDR Y))) (TRNK84 Y (TIMES (TRNK81 (CADR Y) X) (CADDR Y)))) (T (TRNK84
 Y (TRNK89 Y X)))))) (TRNK11 (LAMBDA (Y X N) (COND ((EQUAL (CAR Y) (QUOT
E EQN)) (LIST (QUOTE EQN) (TRNK111 (CADR Y) X N) (TRNK111 (CADDR Y) X N)
NIL)) (T(TRNK111 Y X N)))))
 (TRNK21 (LAMBDA (Y X N) (TRNK211 (TRNK2 Y X N) N)))
(TRNK2 (LAMBDA (Y X N) (COND ((NUMBERP Y) (LIST (QUOTE ATOM) Y (QUOTE (
NIL)))) ((ATOM Y) (LIST (QUOTE ATOM) Y (LIST  (COND ((
EQUAL Y X) 1) (T NIL))) )) ((OR (EQUAL Y X) (NULL(CAR(LAST Y))))Y
) (T (SELECT (CAR Y) ((QUOTE PRD) (LIST (QUOTE PRD)
 (TRNK21 (CADR Y) X (PLUS N (MINUS (TRNK81 (CADDR Y) X)))) (TRNK21 (CADD
R Y) X (PLUS N (MINUS (TRNK81 (CADR Y) X)))) (CADDDR Y))) ((QUOTE PWR) (
LIST (QUOTE PWR) (TRNK21 (CADR Y) X
(DIFFERENCE N (TIMES(SUB1(CADDR Y))(TRNK81(CADR Y)X))))
 )(CADDR Y) (CADDDR Y))) ((QUOTE SUM)
(TRNK2(TRNK8(TRNK7(TRNK24 Y X N)X)X)X N)
) (CONS (CAR Y) (MAPL (CDR Y) (FUNCTION (LAMBDA (U) (TRNK21 U
 X N))))))))) (TRNK24 (LAMBDA (Y X N) (PROG (U V I L S M)
(COND((NULL(CDR(LAST Y)))(RETURN(TRNK25 Y N X))))
(SETQ U(CADR(LAST Y)))
(SETQ V (LIST NIL)) (SETQ I (CADR Y)) (SETQ L (CADDR Y)) (SETQ S (CADDDR
 Y)) A (SETQ M (CDR (SIMP2 (SUBST L I (COPY U)) NIL 0))) (COND ((OR (GRT
P M N) (GRTP L S)) (RETURN (MKP (CDR (SIMP2 (MKPLS V) NIL 0)))))) (SETQ
V (CONS (SUBST L I (NTOL (I) V)) (SETQ L (ADD1 L)) (GO A)))) (TRNKT2 (LA
MBDA (U X Y Z) (COND ((AND (EQUAL U (QUOTE PLS)) (EZEROP Y)) X) ((AND (E
QUAL U (QUOTE PRD)) (EONEP Y)) X) (T (LIST U X Y Z))))) (TRNKT1 (LAMBDA
(X U V Z) (COND ((NULL (CDR V)) (COND ((EQUAL U (QUOTE PLS)) (MKPLS Z))
(T (MKPRD Z)))) ((DEPEND (CAR V) X) (TRNKT2 U (TRNKT (CAR V) X) (TRNKT1
X U (CDR V) Z) (LIST X))) (T (TRNKT1 X U (CDR V) (CONS
(MKCLEANP(CAR V))Z))))))) (
TRNKT(LAMBDA(Y X)(COND((NOT(DEPEND Y X))(MKCLEANP Y))((ATOM Y)Y)
((OR (EQUAL
 (CAR Y) (QUOTE PLS)) (EQUAL (CAR Y) (QUOTE PRD))) (TRNKT1 X (CAR Y) ( C
DR Y) (LIST NIL))) (T (MAPL Y (FUNCTION (LAMBDA (U) (TRNKT U X))))))))
(TRNK82 (LAMBDA (Y X) (SELECT (CAR Y) ((QUOTE PLS) (TRNK84 Y (MIN (TRNK8
1 (CADR Y) X) (TRNK81 (CADDR Y) X)))) ((QUOTE PRD) (TRNK84 Y (PLUS (TRNK
81 (CADR Y) X) (TRNK81 (CADDR Y) X)))) ((QUOTE PWR) (TRNK85 Y X)) ((QUOT
E DRV) (TRNK86 Y X)) ((QUOTE IT() (TRNK87 Y X)) ((QUOTE NAM) (TRNK84 Y (
TRNK81 (NTOL Y) X))) (TRNK88 Y X)))) (TRNK881 (LAMBDA (Y X) (COND ((NULL
 (CDR Y)) NIL) ((EMINUSP (TRNK81 (CAR Y) X)) T) (T (TRNK881 (CDR Y) X)))
)) (TRNK88 (LAMBDA (Y X) (COND ((TRNK881 (CDR Y) X) (TRNK84 Y (TRNK89 Y
X))) (T (TRNK84 Y 0))))) (TRNK87 (LAMBDA (Y X) (COND ((OR (AND (EQUAL (C
ADR Y) X) (EQUAL (CADDR Y) (QUOTE IDF))) (DEPEND (CADDR Y) X) (DEPEND (C
ADDDR Y) X)) (TRNK84 (CONS(QUOTE ASKITG)(CDR Y))(TRNK89 Y X)))
(T (TRNK84 Y(TRNK81 (NTOL Y) X)))
))) (TWOFF (LAMBDA (Y) (COND ((NULL (CDDR Y)) NIL) (T (CONS (CAR Y) (TWO
FF (CDR Y))))))) (TRNK86 (LAMBDA (Y X) (COND ((MEMBER X (TWOFF Y)) (TRNK
84 (CONS(QUOTE ASKDRV)(CDR Y))
     (TRNK89 Y X))) (T (TRNK84 " (TRNK81 (NTOL Y) X)))))) (TRNK89 (LAMBD
A (Y X) (QUESTION (LIST Y) (APPEND (QUOTE (WHAT IS THE LOWEST POWER OF))
 (CONS X (CONS (QUOTE IN) (LIST Y))))))) (TRNK84 (LAMBDA (Y X) (PUTP Y (
LIST  X)))) (PUTP (LAMBDA (Y X) (COND ((NULL (CDR Y)) (LIST
X)) (T (CONS (CAR Y) (PUTP (CDR Y) X))))))) (TRNK83 (LAMBDA
 (Y X) (COND ((AND (OR (EQUAL (CADDDR Y) (QUOTE INF)) (EENUMBERP (CADDDR
 Y))) (NUMBERP (CADDR Y))) (PROG (U) (SETQ U (ERSETQ (CDR (SIMP2 (COPY (
TRNK831 (NTOL Y) X (CADR Y))) NIL 0)))) (COND ((NULL U) (SETQ (LIST
  (TRNK8311 Y X (CADR Y)))))) (SETQ U (MKP (CAR U))) (COND ((AND (JUSTDE
PEND U (CADR Y)) (INCFNP U (CADR Y))) (RETURN(TRNK841(TRNK84 Y (CDR(SIMP
2 (COPY (SUBST (CADDR Y) (CADR Y) U)) NIL 0))) U))))
(RETURN (TRNK84 Y (TRNK89 Y X))))) (T (TRNK84 Y (TRNK89 Y X))
)))) (GETL (LAMBDA (Y X) (COND ((NULL Y) NIL) ((EQUAL (CAAR Y) X) (CDAR
Y)) (T (GETL (CDR Y) X)))))) (TRNK81 (LAMBDA (Y X) (COND ((EQUAL Y X) 1)
((ATOM Y)0)((NULL(CAR(LAST Y)))0)(T(CAR(LAST Y))))))(TRNK831(LAMBDA(
Y X I) (COND ((NOT (DEPEND Y X))0)((NOT(DEPEND(MKP Y)I))(TRNK81 (TRNK8 Y
 X) X)) (T (SELECT (CAR Y) ((QUOTE PRD) (LIST (QUOTE PLS) (TRNK831 (CADR
 Y) X I) (TRNK831 (CADDR Y) X I) NIL)) ((QUOTE PLS) (LIST (QUOTE MIN) (T
RNK831 (CADR Y) X I) (TRNK831 (CADDR Y) X I) NIL)) ((QUOTE PWR) (COND ((
AND(NOT(DEPEND(MKP(CADR Y))I))(NOT (DEPEND (CADDR Y) X))) (LIST (QUOTE P
RD) (TRNK831 (CADR Y) X I) (CADDR Y) NIL)) (T (TRNK8311 Y X I)))) ((QUOT
E NAM) 0) (TRNK8311 Y X I)))))) (TRNK8 (LAMBDA (Y X) (COND ((ATOM Y) Y)
((EQUAL Y X) Y) ((NOT (DEPEND Y X))(TRNK84 Y NIL))((EQUAL (CAR Y) (QUOTE
 SUM)) (TRNK83 Y X)) (T (TRNK82 (CONS (CAR Y) (MAPL (CDR Y) (FUNCTION (L
AMBDA (U) (TRNK8 U X))))) X)))))
(TRNK8311 (LAMBDA
      (Y X I) (QUESTION (LIST Y) (APPEND (QUOTE (RETURN THE LOWEST POWER
OF)) (CONS X (APPEND (QUOTE (AS A FUNCTION OF)) (LIST I))))))) (JUSTDEPE
ND (LAMBDA (Y X) (COND ((OR (EENUMBERP Y) (EQUAL X Y)) T) ((ATOM Y) NIL)
 (T (NULL (STRIKE (LAST Y) (FUNCTION (LAMBDA (U) (EQUAL U X))))))))) (IN
CFNP (LAMBDA (Y X) (COND ((EQUAL Y X) T) ((NOT (DEPEND Y X)) NIL) ((MEMB
ER (CAR Y) (QUOTE (PLS MIN))) (EAND (MAPL (CDR Y) (FUNCTION (LAMBDA (U)
(COND ((DEPEND U X) (INCFNP U X)) (T T))))))) ((MEMBER (CAR Y) (QUOTE (P
RD PWR))) (EAND (MAPL (CDR Y) (FUNCTION (LAMBDA (U) (COND ((DEPEND U X)
(INCFNP U X)) (T (PLSFNP U)))))))) (T NIL)))) (PLSFNP (LAMBDA (Y) (AND (
NUMBERP Y) (NOT (EMINUSP Y))))) (EAND (LAMBDA (Y) (COND ((NULL (CDR Y))
T) ((CAR Y) (EAND (CDR Y))) (T NIL))))))
DEFINE((
(TRNK841(LAMBDA(Y U)(COND
((NULL(CDR Y))(LIST(LIST(CAAR Y)U)))
(T(CONS(CAR Y)(TRNK841(CDR Y)U))) )))
(TRNK211(LAMBDA(Y X)(COND
((NULL(CDR Y))(LIST(CONS X(CAR Y))))
(T(CONS(CAR Y)(TRNK211(CDR Y)X))) )))
(MKCLEANP(LAMBDA(X)(COND
((ATOM X)X)
((NULL(CDR X))(LIST NIL))
(T(CONS(MKCLEANP(CAR X))(MKCLEANP(CDR X)))))))
))
DEFINE((
(TRNK25(LAMBDA(Y N X)(QUESTION(LIST Y)
(APPEND(QUOTE(WHAT IS THE EXPANSION OF))(LIST
 Y(QUOTE IN)X(QUOTE TO)(QUOTE POWER)N)))))
))
```

TRNK9 LISP

```
DEFINE
(((TRNK123 (LAMBDA (Y N X) (PROG NIL
 (FILEWRITE (QUOTE LISPTR) (QUOTE LISP) (TRNK12 Y N X))
(PRIN1 DOLLAR)(PRINT(QUOTE(ABSORBE)))(RETURN NIL))
)) (EAND (LAMBDA (Y) (COND ((NULL (CDR Y)) T) ((CAR Y) (EAND (CDR Y))) (
T NIL)))) (TRNK12 (LAMBDA (Y N X) (COND ((EQUAL (CAR Y) (QUOTE EQN)) (LI
ST (CAR Y) (TRNK12 (CADR Y) N X) (TRNK12 (CADDR Y) N X) NIL)) (T (MKPLS
(TRNK121 (TRNK9 Y X NIL) N X)))))) (TRNK63 (LAMBDA (Y U V S R)
 (TRNK6 (REVERSE (TRNK10 (REVERSE (SRPWR Y U R
 (MINUS S)))))))) (TRNK62 (LAMBDA (X Y) (COND ((NULL X) (LIST NIL)) ((O
R (EZEROP (CAR X)) (EZEROP (CAR Y))) (TRNK62 (CDR X) (CDR Y))) (T (CONS
(EPRD (CAR X) (CAR Y)) (TRNK62 (CDR X) (CDR Y))))))) (TRNK61 (LAMBDA (W
V U N) (COND ((GREATERP N (LENGTH U)) W) (T (TRNK61 (CONS (EPRD V (MKPLS
 (TRNK62 W U))) W) V U (ADD1 N)))))) (TRNK6 (LAMBDA (X) (PROG (U V) (CON
D ((NULL X) (RETURN (QUESTION NIL (QUOTE (DIVISION BY ZERO)))))) (SETQ U
 (REVERSE X)) (SETQ V (EDVD 1 (CAR U))) (RETURN (TRNK61 (LIST V) (ENEG V
) (CDR U) 1))))) (GETL (LAMBDA (Y X) (COND ((NULL Y) NIL) ((EQUAL (CAAR
Y) X) (CDAR Y)) (T (GETL (CDR Y) X))))) (TRNK5 (LAMBDA (X Y) (COND ((OR
(NULL X) (NULL Y)) NIL) (T (REVERSE (TRNK2222 (TRNK51 (SUB1 (LENGTH Y))
(REVERSE X)) Y)))))) (TRNK2222 (LAMBDA (X Y) (COND ((NOT (EZEROP (CAR X)
)) (LIST (MKPLS (TRNK3 X Y)))) (T (CONS (MKPLS (TRNK3 X Y)) (TRNK2222 (C
DR X) Y)))))) (TWOFF (LAMBDA (Y) (COND ((NULL (CDDR Y)) NIL) (T (CONS (C
AR Y) (TWOFF (CDR Y))))))) (TRNK1 (LAMBDA (X Y) (COND ((NULL X) Y) ((NUL
L Y) X) (T (CONS (EPLS (CAR X) (CAR Y)) (TRNK1 (CDR X) (CDR Y))))))) (TR
NK51 (LAMBDA (N X) (COND ((ZEROP N) X) ((MINUSP N) NIL) (T (CONS 0 (TRNK
51 (SUB1 N) X)))))) (TRNK3 (LAMBDA (X Y) (COND ((OR (NULL X) (NULL Y)) (
LIST NIL)) ((OR (EZEROP (CAR Y)) (EZEROP (CAR X))) (TRNK3 (CDR X) (CDR Y
))) (T (CONS (EPRD (CAR X) (CAR Y)) (TRNK3 (CDR X) (CDR Y))))))) (SRPWR
(LAMBDA (X R N S) (COND ((GRTP (LENGTH X) (ADD1 R)) (SRPWR2 N (TAKE X (A
DD1 R)) (TAKE (REVERSE X) (PLUS (LENGTH X) (MINUS (ADD1 R)))) (PLUS (ADD
1 R) (MINUS (LENGTH X))) R S)) (T (SRPWR3 N (TAKE X (ADD1 R)) R S))))) (
SRPWR1 (LAMBDA (P M R V S2 A) (COND ((ZEROP M) 1) ((OR (LSSP (PLUS P (MI
NUS (TIMES (ADD1 R) V))) (TIMES S2 (PLUS M (MINUS (ADD1 R))))) (NOT (LSS
P R M))) (SRPWR11 P M R V S2 A)) (T (EPLS (SRPWR11 P M R V S2 A) (SRPWR1
 P M (ADD1 R) V S2 A))))))) (SRPWR2 (LAMBDA (N X Y S2 S1 S) (PROG (J U M
P) (SETQ J (SUB1 (TIMES S2 S))) A (SETQ J (ADD1 J)) (COND ((GRTP J N) (R
ETURN U))) (SETQ U (CONS 0 U)) (SETQ M (SUB1 (COND ((MINUSP J) (COND ((Z
EROP (REMAINDER J S2)) (QUOTIENT J S2)) (T (ADD1 (QUOTIENT J S2))))) (T
0)))) B (SETQ M (ADD1 M)) (COND ((GRTP M (QUOTIENT (PLUS (TIMES S1 S) (M
INUS J)) (PLUS  0.1E1 S1))) (GO D))) (SETQ P (ADD1 (MIN (MINUS (TIMES M
S2)) (PLUS (MINUS J) (TIMES S1 (PLUS S (MINUS M))))))) C (SETQ P (SUB1 P
)) (COND ((OR (LSSP P M) (LSSP P (MINUS J))) (GO B))) (SETQ U (CONS (EPL
S (CAR U) (EPRD (SRPWR1 P M (MAX 0 (PLUS P (TIMES (ADD1 S2) M))) S2 1 Y)
 (SRPWR1 (PLUS J P) (PLUS S (MINUS M)) (MAX 0 (PLUS J P (MINUS (TIMES (P
LUS S1 -1) (PLUS S (MINUS M)))))) S1 0 X))) (CDR U))) (GO C) D (SETQ U (
CONS (EPRD (FTLEXP S) (CAR U)) (CDR U))) (GO A)))) (SRPWR3 (LAMBDA (N A
R S) (PROG (M U Q) (SETQ M 0) B (COND ((GRTP M N) (RETURN U))) (SETQ U (
CONS (EPRD (FTLEXP S) (SRPWR1 M S (MAX 0 (PLUS M (MINUS (TIMES (SUB1 R)
S))))) R 0 A)) U)) (SETQ M (ADD1 M)) (GO B)))) (SRPWR11 (LAMBDA (P M R V
S2 A) (COND ((NULL (CDR A)) (EDVD (EPWR (CAR A) M) (FTLEXP R))) (T (EPRD
 (EDVD (EPWR (CAR A) R) (FTLEXP R)) (SRPWR1 (PLUS P (MINUS (TIMES R V)))
 (PLUS M (MINUS R)) (MAX 0 (PLUS P (MINUS (TIMES 2 R)) (TIMES M (PLUS 2
(MINUS V))))) (SUB1 V) S2 (CDR A))))))) (TAKE (LAMBDA (X N) (COND ((ZERO
P N) NIL) ((NULL X) (CONS 0 (TAKE NIL (SUB1 N)))) (T (CONS (CAR X) (TAKE
 (CDR X) (SUB1 N))))))) (TRNK10 (LAMBDA (X) (COND ((EZEROP (CAR X)) (TRN
K10 (CDR X))) (T X))))) (TRNK13 (LAMBDA (Y N X) (PROG (U V R) (SETQ R 0)
(SETQ U (MKP Y)) A (SETQ V (CONS (EDVD (TRNK131 U X) (FTLEXP R)) V)) (CO
ND ((EQUAL R N) (RETURN V))) (SETQ R (ADD1 R)) (SETQ U (MKP (DIFF2 U X))
) (GO A)))) (TRNK131 (LAMBDA (Y X) (COND ((NOT (DEPEND Y X)) (COPY Y)) (
(EQUAL Y X) 0) ((EQUAL (CAR Y) (QUOTE DRV)) (LIST (QUOTE EVL) X 0 (COPY
Y) NIL)) (T (CONS (CAR Y) (MAPL (CDR Y) (FUNCTION (LAMBDA (U) (TRNK131 U
 X))))))))) (TRNK121 (LAMBDA (Y N X) (COND ((NULL Y) (LIST NIL)) (T (CON
S (EPRD (CAR Y) (EPWR X N)) (TRNK121 (CDR Y) (SUB1 N) X)))))) (TRNK9 (LA
MBDA(Y X Z)(COND
((OR(ATOM Y)(NULL(CADR(LAST Y))))(COND((EQUAL(CAR Y)(QUOTE ATOM
)) (COND ((EZEROP (CADR Y)) NIL) (T (TRNK51 (TRNK91 Y) (LIST (CADR Y)))))
)) (T (TRNK51 (TRNK91 Y) (LIST Y))))) (T (SELECT (CAR Y)
((QUOTE ASKDRV)(TRNK25 Y(CONS(QUOTE DRV)(CDR Y))(TRNK91 Y)X))
((QUOTE ASKITG)(TRNK25(CONS(QUOTE ITG)(CDR Y))(TRNK91 Y)X))
((QUOTE PLS) (TRNK1
     (TRNK9 (CADR Y) X Z) (TRNK9 (CADDR Y) X Z))) ((QUOTE PRD) (TRNKS (T
RNK9 (CADR Y) X Z) (TRNK9 (CADDR Y) X Z))) ((QUOTE PWR) (COND ((GRTP (TI
MES (CADDR Y) (TRNK92 (CADR Y))) (TRNK91 Y)) NIL) ((EMINUSP (CADDR Y)) (
TRNK63 (TRNK9 (CADR Y) X Z) (TRNK91 (CADR Y)) (TRNK91 Y) (CADDR Y) (PLUS
 (TRNK91 Y)(TIMES 2(MINUS (CADDR Y)) (TRNK92 (CADR Y))))))) (T (REVERSE (
TRNK10 (REVERSE (SRPWR (TRNK9 (CADR Y) X Z) (TRNK91 (CADR Y)) (TRNK91 Y)
 (CADDR Y))))))))) ((QUOTE ITG) (TRNK93 (TRNK9 (NTOL Y) X Z) (TWOFF Y)))
((QUOTE DRV) (TRNK93 (TRNK9 (NTOL Y) X Z) (TWOFF Y))) ((QUOTE ATOM) (TRN
K51 (SUB1 (TRNK91 Y)) (LIST 1))) ((QUOTE NAM) (TRNK95 Z (COND ((NOT (TRN
K94 Y)) (ATOMOFF (APPEND (TWOFF Y) (LIST (TRNK96 (NTOL Y) X) NIL)))) (T
(ATOMOFF Y))) (TRNK91 Y) X)) (TRNK95 Z (COND ((NOT (TRNK94 Y)) (TRNK96 Y
 X)) (T (ATOMOFF Y))) (TRNK91 Y) X))))) (ATOMOFF (LAMBDA (Y) (COND ((AT
OM Y) Y) ((EQUAL (CAR Y) (QUOTE ATOM)) (CADR Y)) (T (CONS (CAR Y) (MAPL
(CDR Y) (FUNCTION ATOMOFF))))))) (TRNK91 (LAMBDA (Y) (CAR  (LAST Y)
))) (TRNK92(LAMBDA(Y)(COND
((NULL(CADR(LAST Y)))0)
(T(CADR(LAST Y)))    ))
)
 (TRNK93 (LAMBDA (Y X) (COND ((NULL Y) NIL) ((EZEROP (CAR Y)) (CONS 0 (T
RNK93 (CDR Y) X))) (T (CONS (APPEND X (LIST (CAR Y) NIL)) (TRNK93 (CDR Y
) X)))))) (TRNK94(LAMBDA(X)(COND
((OR(ATOM X)
(NULL(LAST X))
(NULL(CADR(LAST X)
)) (EQUAL (CAR X) (QUOTE ATOM))) T) ((MINUSP (CADR(LAST X))
) NIL) (T (EAND (MAPL X (FUNCTION TRNK94)))))))) (TRNK95
(LAMBDA (Z U N X) (COND (Z U) (T (TRNK13 U N X))))) (TRNK96 (LAMBDA (Y X
) (CONS (CAR Y) (TRNK961 (CDR Y) N X)))) (TRNK961 (LAMBDA (Y N X) (COND
((NULL (CDR Y)) Y) (T (CONS (MKPLS (TRNK121 (TRNK9 (CAR Y) X T) N X)) (T
RNK961 (CDR Y) N X))))))))
DEFINE((
(TRNK25(LAMBDA(Y N X)(CDR(REVERSE(CDR(REVERSE(QUESTION(LIST(ATOMOFF Y))
(APPEND(QUOTE(WHAT IS THE SET OF COEFFICIENTS IN THE EXPANSION OF))(LIST
 (ATOMOFF Y)(QUOTE IN)X(QUOTE TO)(QUOTE POWER)N)))))))))
))
DEFINE((
(QUESTION(LAMBDA(X Y)(PROG(V)
(FILEWRITE(QUOTE LISPTR)(QUOTE LISP)(QUOTE E))
(FILEAPND(QUOTE LISPTR)(QUOTE LISP)(LIST(QUOTE PROG) NIL
(LIST(QUOTE FILEWRITE)(QUOTE(QUOTE LISPTR))(QUOTE(QUOTE RETURN)))
(LIST(QUOTE QUESTION)(LIST(QUOTE QUOTE)X) (LIST(QUOTE QUOTE)Y)))
(QUOTE(PRIN1 DOLLAR))
(QUOTE(PRINT(QUOTE(ABSORBE))))   )))
(FILEAPND(QUOTE LISPTR)(QUOTE LISP)(QUOTE CHAIN))
(FILEAPND(QUOTE LISPTR)(QUOTE LISP)(QUOTE(((RESUME G002)))))
(CHAIN(QUOTE(
(SAVE G002 T)
(RESUME LISPB LISPTR)
```

```
)))
(FILESEEK(QUOTE LISPTR)(QUOTE RETURN))
(SETQ U(READ))
(FILEENDRD(QUOTE LISPTR)(QUOTE RETURN))
(RETURN U))))
))
```

```
                 TRNK9 COMP
SPECIAL((X N))
COMPILE((
EAND TRNK12 TRNK63 TRNK62 TRNK61 TRNK6 GETL TRNK5
TRNK2222 TWOFF TRNK1 TRNK51 TRNK3 SRPWR SRPWR1 SRPWR2
SRPWR3 SRPWR11 TAKE TRNK10 TRNK13 TRNK131 TRNK121 TRNK9
ATOMOFF TRNK91 TRNK92 TRNK93 TRNK94 TRNK95 TRNK96 TRNK961))
UNSPECIAL((X N))
STOP
```

BIBLIOGRAPHY

J. Sammet has prepared an exhaustive bibliography. Therefore, only a few references appropriate to each chapter will be given here. Sammett's bibliography is listed as the first reference.

Chapter 1

1. Sammet , J. "An Annotated Descriptor Based Bibliography on the Use of Computer for Non-Numerical Mathematics", Computing Reviews (July, 1966).

2. Licklider, J.C.R. "Man-Computer Symbiosis", I.R.E. Human Factors, (March, 1960).

3. Hart, T.P. "A Heuristic Program for the Solution of Electrical Networks", S.M. Thesis, , M.I.T. (1961) E.E. Dept.

4. Wooldridge, D. Jr. "An Algebraic Simplify Program in LISP", Stanford Art.Intell. Memo No. 11, (1963)

5. Korsvold, K. "An On Line Algebraic Simplify Program", Stanford Art.Intell. Memo No. 37 (Nov.1965).

6. Fenichel, R. "Famous", Ph.D. Thesis, Harvard, 1966.

7. Slagle, J.R. "A Heuristic Program that Solves Symbolic Integration Problems in Freshman Calculus", M.I.T. Lincoln Lab.Report 5G0001, (1961).

8. Moses, J. "Symbolic Integration", M.I.T. Art. Intel. Memo No. 97 (June 1966).

9. Brown, W.S. "The Alpak System for Non-numerical Algebra on a Digital Computer", Bell System Technical Journal (Sept.1963).

10. Collins, G.E. "PM, A System for Polynomial Manipulation", I.B.M. Research Paper RC-1526 (Jan.1966)

11. Sammet , J.E. "Introduction to FORMAC", I.E.E.E. Transactions on Electronic Computers , Vol. EC-13, No. 4 (Aug.1964).

12. Perlis, A., Iturriaga, R. and Standish, T.A. "A Definition of Formula Algol", Carnegie Inst. of Tech., (March 1966).

13. Shaw, J.C. "JOSS: A Designer's View of an Experimental On-Line Computing System", Fall Joint Computer Conference (1964).

14. Culler, G.J., "An On-Line Computing Center for Scientific Problems", Thompson Ramo Wooldridge Inc., Report M19-343, (1963).

15. Kaplow, R., Strong, S., and Brackett, J. "MAP: A System for On-Line Mathematical Analysis", M.I.T. Project MAC Report MAC-TR-24 (1966).

16. Engleman, C. "MATHLAB: A program for On-Line Machine Assistance in Symbolic Computations", Proceedings FJCC (Nov. 1965).

17. Engleman, C. "Rational Functions in MATHLAB" Proc of IFIP Working Conference on Symbol Manipulation Languages, Pisa, Italy (1966).

18. Richardson, D. Doc. Thesis, U. of Bristol, Bristol, England.

19. Levin, M. LISP 1.5 Programmer's Manual, M.I.T. Press, Cambridge, Mass.

Chapter II

1. Lighthill, M.J. "A technique for Rendering Approximate Solutions to Physical Problems Uniformly Valid", Phil. Mag. (7) 40, 1179-1201.

2. M.I.T. Mathematics Dept., Course 18.655, Notes and References.

3. Garber, H.N. "A Class of Queueing Problems," Interim Tech. Report No. 13, M.I.T. Center for Operations Research (1959).

4. Draper, J.S. "Plasma Accelerator Electrode Boundary Layers", M.S. Thesis, M.I.T. Aero (1964).

5. Kerrebrock, J.L. "Electrode Boundary Layers, in Direct-Current Plasma Accelerators", J. of the Aerospace Sciences, 28 no. 8, (August 1962) 633-643.

6. Tobey, R.G. "Eliminating Monotonous Mathematics with FORMAC", Comm. of the ACM, 9, No.10 (Oct.1966).

Chapter III

1. Teitleman, W. "PILOT - A Step Toward Man-Computer Symbiosis", Ph.D. Thesis, M.I.T. (Sept. 1966).

2. Bobrow, D.G. "Natural Language Input for a Computer Problem Solving System", Ph.D. Thesis, M.I.T. (Sept. 1964).

3. Feigenbaum, E.A. and Feldman, J. "Computers and Thought", McGraw-Hill, New York, (1963).

4. Maurer, W.D. "Computer Experiments in Finite Algebra," M.I.T. Project MAC memo, MAC-M-246 (1965).

5. Hearn, A. Computation of Algebraic Properties of Elementary Particle Reactions Using a Digital Computer," Dept. of Physics, Stanford, ITP-196 (1966).

Chapter V

1. Ralston, A. and Wilf, H.S. "Mathematical Methods for Digital Computers", Wiley, New York (1960).

2. Van Trees, H.L. "Synthesis of Optimum Nonlinear Control Systems", M.I.T. Press, Cambridge (1962).

3. Newton, G.C., Gould, L.A., Kaiser, J.F. "Analytical Design of Linear Feedback Controls", Wiley, New York, (1957).

4. Crandall, S.H. "Engineering Analysis. A Survey of Numerical Procedures", McGraw-Hill, New York (1965).

5. Jolley, L.B.W. " Handbook of Summable Series ", (1925).

6. Cayley, A. "On a Soluable Quintic Equation", Amer.J.Math (1892).

7. Runge, C. "Ueber die Zerlegung ganzer ganzzahliger Functionen in irreductible Factoren", J.für die Reine und Angewandte Mathematik (1886).

8. Moses, J. "Solutions of Systems of Polynomial Equations by Elimination", Comm. of the ACM., 9, No. 8 (Aug. 1966).

Chapter VI

1. Levin, M. LISP 1.5 Programmer's Manual, M.I.T. Press Cambridge.

2. Berkeley, E.C. and Bobrow, D.G. "The Programming Language LISP: Its Operation and Applications", Information International Inc., Cambridge (1964).

Chapter VII

1. Birkhoff G and MacLane, S. "A Survey of Modern Algebra", Macmillan (1960).

2. Albert, A. Adrian "Fundamental Concepts of Higher Algebra", Phoenix Science Series, U. of Chicago Press (1963).

3. Bartee, T. and Schneider, D. "Computation with finite fields", Information and Control, 6 (1963).

4. Garner, H.L. "The residue number system", IRE Trans. on Electronic Computers, 8 (June 1959).

5. Fraenkel, A.S. "The use of index calculus and Mersenne primes for design of a high-speed digital multiplier", JACM, (Jan. 1961).

6. Ankeny, N.C. M.I.T. Mathmatics Department, discussion.

7. Richardson, D. Doctoral Thesis, U. of Bristol, Bristol, England.

8. Spitzbart, A. and Bardell, R.A. "College Algebra and Plane Trigonometry," Adison-Wesley, Cambridge (1955).

Chapter IX

1. Minsky, M.L. "Mathscope: Part I." Artificial Intelligence Memo 61.

2. Krakauer, L.J. "Syntax and Display of Printed Format Mathmatical Formulas", M.I.T., MS Thesis, (1964).

3. The William Byrd Press. "Mathematics in Type", The William Byrd Press, Richmond, Virginia, (1954).

4. Wells, Mark B. "MADCAP: A Scientific Compiler for a Displayed Formula Textbook Language". Communications of the ACM, 4, No. 1 (January 1961).

5. Klerer, M. and May, J. "An Experiment in a User Oriented Computer System", Communications of the ACM, 7, No. 5 (May 1964).

Chapter X

1. Floyd, R. W. "Syntatic Analysis and Operator Precedence", JACM, 10, No. 3 (July 1963).

Chapter XI

1. Teitelman, W. "New Methods for Real-Time Recognition of Hand-Drawn Characters", MS Thesis, M.I.T. (1963).

2. Bobrow, D.G. "Syntactic Analysis of English by Computer - A Survey", Proc. FJCC, Spartan Press, Baltimore, Md., (1963).

3. Ross, D.T. and Feldman, C.G. "Verbal and Graphical Language for the AED System; A Progress Report," MIT Project MAC Tech Report, MAC-TR-4.

4. Klerer, M. Unpublished notes at Hudson Laboratories of Columbia University (1965).

Chapter XIV

1. Programmed Data Processor - 6 Handbook, Digital Equipment Corporation, Maynard, Massachusetts.

2. Lang, C. "New B-Core System for Programming the ESL Display Console", MIT ESL Memo 9442-M-122/MAC-M-216, (April, 1965).

Biographical Note on the Author

William A. Martin was born in Oklahoma City on May 18, 1938, and graduated from Northwest Classen Highschool in 1956. He entered M.I.T. as a National Merit Scholar and received a B.S. and M.S. in Electrical Engineering in 1962. During this period he was elected to Tau Beta Pi, Eta Kappa Nu, and Sigma Xi. He was a member of the Beta Theta Pi social fraternity and captain of the wrestling team. He was a teaching assistant in the Electronic Circuits and Signals Laboratory and a Research Assistant in the Center for Operations Research.

Mr. Martin has held summer positions with electronics firms in England, Germany, and Switzerland and has been employed by the MITRE Corporation, Information International Incorporated, and IBM.

He published:

Description and Control of Single Lane Tunnel Traffic Flow, Research Report No. 3 of the Center for Operations Research, M.I.T. (1963).