MAC-TR-47 (THESIS)

SYMBOLIC INTEGRATION

by

Joel Moses

December 1967

Project MAC

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Massachusetts Institute of Technology
Project MAC
545 Technology Square
Cambridge, Massachusetts
02139

Work reported herein was supported in part by Project MAC, an M.I.T. research project sponsored by the Advanced Research Projects Agency, Department of Defense, under Office of Naval Research Contract Nonr-4102(01). Reproduction of this report, in whole or in part, is permitted for any purpose of the United States Government.

AUTHOR'S NOTE

A condensation of this and other relevant work, in the field of algebraic manipulation, is being written in collaboration with William A. Martin of the Department of Electrical Engineering and will be published next year by M.I.T. Press.

SYMBOLIC INTEGRATION

by

Joel Moses

B.A., Columbia College
(1962)

M.A., Graduate Faculties, Columbia University
(1963)

SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE
DEGREE OF DOCTOR OF
PHILOSOPHY

at the

MASSACHUSETTS INSTITUTE OF
TECHNOLOGY

September, 1967

Signature of Author . . . . . Joel Moses
Department of Mathematics, September 1, 1967

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

Accepted by . . . . . M. Arthur Levinson
Chairman, Departmental Committee
on Graduate Students

SYMBOLIC INTEGRATION

by

Joel Moses

Submitted to the Department of Mathematics on September 1, 1967 in partial fulfillment of the requirements for the degree of Doctor of Philosophy

ABSTRACT

SIN and SOLDIER are heuristic programs written in LISP which solve symbolic integration problems. SIN (Symbolic INtegrator) solves indefinite integration problems at the difficulty approaching those in the larger integral tables. SIN contains several more methods than are used in the previous symbolic integration program SAINT, and solves most of the problems attempted by SAINT in less than one second. SOLDIER (SOLution of Ordinary DIfferential Equations Routine) solves first order, first degree ordinary differential equations at the level of a good college sophomore and at an average of about five seconds per problem attempted. The differences in philosophy and operation between SAINT and SIN are described, and suggestions for extending the work presented are made.

Thesis Supervisor: Marvin L. Minsky

Title: Professor of Electrical Engineering

ACKNOWLEDGMENTS

The work reported herein was supported in part by Project MAC, an MIT research program sponsored by the Advanced Research Projects Agency, Department of Defense, under Office of Naval Research, contract Number Nonr-4102(01), in part by the Joint Services Electronics Program (contract DP 28-043-AMC-02536 (E)), the National Science Foundation (Grant GP-2495), and the National Aeronautics and Space Administration (Grant NsG-496). Reproduction in whole or in part is permitted for any purpose of the United States Government.

I wish to thank Professor Marvin Minsky, the supervisor of this thesis, and Professors Seymour Papert and Joseph Weizenbaum, the other members of my thesis committee, for their criticism and guidance of this work. I also wish to thank William Martin for many constructive discussions and suggestions. To Carl Engelman, Michael Manove, and Stephen Bloom goes my gratitude for the use of their program, and to Ima, Laila and Sandy it goes for having to read my handwriting.

TABLE OF CONTENTS

	Abstract	1
	Acknowledgments	2
	Table of Contents	3
	Dedication	4
Chapter 1	Introduction	5
Chapter 2	How SIN differs from SAINT	11
Chapter 3	SCHATCHEN - A Matching Program for Algebraic Expressions	22
Chapter 4	SIN - The Symbolic Integrator	62
Chapter 5	The Edge Heuristic	107
Chapter 6	Solution of Ordinary Differential Equations	124
Chapter 7	Conclusions and Suggestions for Further Work	140
Appendix A	ITALU - An Integral Table Look-Up	153
Appendix B	Recursively Unsolvable Results in Integration	160
Appendix C	SIN's Performance on SAINT's Problems	172
Appendix D	Solution of Problems Proposed by McIntosh	176
Appendix E	An Experiment with SOLDIER	180
Appendix F	Listings of the Programs	193
	Bibliography	262
	Biography of the Author	267

To the descendants of the Maharal
who are endeavoring to build a Golem.

Chapter 1

Introduction

In the last few years there has been a surge of activity on the design of algebraic manipulation systems*. Algebraic manipulation systems are computer based systems which facilitate the handling of algebraic and analytic expressions. One of the oft stated capabilities desired of such systems is an ability to perform symbolic integration. Besides the obvious value of such a capability in symbolic calculations there is the possibility of employing it as an adjunct to numerical integration programs for functions which involve parameters. In such cases a single accurate symbolic integration is likely to be preferable to numerical integrations taken over the range of values of the parameters. Another reason for the interest in symbolic integration programs is the fact that the ease with which such a program could be written in a proposed language for algebraic manipulation has become an informal test of the power of that language. Yet the only previously announced symbolic integration program with any claim to generality is SAINT (Symbolic Automatic INTegrator), written as a doctoral dissertation by Slagle in 1961 [58]. Slagle described SAINT as being as powerful as a good freshman calculus student. Thus the unmodified SAINT program does not appear powerful enough to warrant

*For a survey of the field of algebraic manipulation see Sammet [55]. For a bibliography of work in the field up to 1966 see Sammet [56].

its use in a practical algebraic manipulation system. In 1964 a program which integrates rational functions was written for the MATHLAB project by Manove, Bloom, and Engelman of the MITRE Corporation [36]. This program filled an important gap in the capabilities of SAINT. By using such a program it appeared possible to write a more powerful integration program than SAINT. Furthermore it seemed that programs which solve ordinary differential equations at least as well as sophomore college students (and a good deal faster than such students) could also be written. Such programs became the goals of our research.

We used the rational function package of MATHLAB in writing a second symbolic integration program called SIN (Symbolic INtegrator). SIN, in turn, we used to write a program which solves first order, first degree ordinary differential equations. This program is called SOLDIER (SOLution of DIfferential Equations Routine). SIN and SOLDIER are both written in LISP [34], [20] for the CTSS system at Project MAC [11]. These experiments in symbolic integration are the principal subjects of this thesis. We believe these programs to possess sufficient power and efficiency that they could be effectively used in a practical on-line algebraic manipulation system.

In order to clarify the domain of applicability of our programs and in order to indicate the power of the present versions of SIN and SOLDIER, we present below two examples of problems solved by each program. The solutions that these programs obtain to the four problems can be found in Chapters 4 and 6.

$$\int \frac{\sqrt{A^2+B^2\sin^2 x}}{\sin x}\,dx$$

$$\int (1+2x^2)\, e^{x^2}\,dx$$

$$(2xy+5x+1)y'+y^2=0$$

$$(y+x-1)y'-y+2x+3=0$$

Problems solved by SIN and SOLDIER

Figure 1

Although the capabilities of SAINT are quite impressive, we found compelling reasons for taking, in SIN, a substantially different approach. The most fundamental difference between SIN and SAINT is in the organization of the programs. SAINT utilizes a tree search as its main organizational device. Slagle compares the behavior of SAINT to that of freshman calculus students. We sought an organizational model which behaved like our conception of the behavior of an expert human integrator. This model was supposed to determine the methods needed to solve a problem quite quickly. A discussion of the approach taken in SIN is given in Chapter 2.

SAINT utilizes a matching program for algebraic expressions called Elinst (ELementary INSTance). We desired a program which was more closely organized as an interpreter for a pattern matching language. This program. called SCHATCHEN, is a service routine employed throughout SIN and SOLDIER. The power of SCHATCHEN greatly simplified the problem of writing an algebraic simplification program, called SCHVUOS. SCHATCHEN and SCHVUOS are described in Chapter 3.

Chapter 4 contains a detailed description of SIN and its methods. A comparison between methods used in SAINT and SIN is made. It is noted that SIN contains several methods not included in SAINT. Among these is a decision procedure for a set of integration problems. Thus SIN is able to determine that $\int e^{x^2} dx$ and $\int \frac{e^x}{x} dx$ are not integrable in closed form.

In Chapter 5 we introduce the Edge (EDucational GuEss) heuristic. The Edge heuristic is based on the Liouville theory of integration. In this theory it is shown that if a function is integrable in closed form, then the form of the integral can be deduced up to certain coefficients. A program which employs the Edge heuristic, called Edge, uses a simple analysis to guess at the form of the integral and then it attempts to obtain the coefficients. Edge is a nontraditional integration method and one that we believe is the first in a line of very powerful methods.

The methods and organization of SOLDIER are introduced in Chapter 6. The area of nonlinear first order differential equations is much more difficult than just integration. Thus we were hardly surprised at not being able to find a concept analogous to the Edge heuristic of SIN. Nonetheless the power of the current version of SOLDIER is comparable to that of a sophomore student in an ordinary differential equations course.

The appendices contain results of experiments performed with SIN and SOLDIER and a report on some other work not directly concerned with these programs.

Many people probably believe that the cheapest way to obtain an integration capability would be to design an integral table look-up program. While we do not espouse this course of action, we did experiment with such a program (called ITALU). Appendix A describes this program.

Richardson has recently obtained a recursive unsolvability result in integration wich has aroused great interest [52]. We describe this theorem and present some of our own related results which involve nonlinear differential equations in Appendix B.

SAINT was asked to solve 86 problems. Of these it solved 84 in an average time of 2.4 minutes. SIN solved all 86 problems with solution times which were frequently more than two orders of magnitude faster than SAINT. SIN solved the other two problems by using integration methods not available in SAINT. The fact that SIN was compiled and that SAINT was run interpretively accounted for most of the gain in speed. Results and further interpretations of this experiment are given in Appendix C.

A physicist, Harold McIntosh, used an integral table to solve eleven fairly difficult integration problems. SIN, after some prodding, solved these problems and found some minor errors in Professor McIntosh's answers. This experiment is described in Appendix D.

In order to test the effectiveness of SOLDIER we asked it to solve 76 problems taken out of a differential equations text. SOLDIER solved 67 of these problems cleanly with an average time of

about five seconds. One of these solutions indicated a misprint in the solution given in the text. This experiment is described in Appendix E.

With the exception of Chapter 7 which presents conclusions and suggestions for further work the following chapters are fairly self contained. Thus those who are only interested in algebraic manipulation can reasonably ignore Chapter 2. Those interested in AI may wish to ignore the higher numbered chapters.

CHAPTER 2

HOW SIN DIFFERS FROM SAINT

Introduction

In this chapter we discuss in broad terms the organizational differences between SIN and SAINT. SAINT employs rather loose progress constraints in generating subproblems, and obtains a solution through a tree search. SIN relies on a much tighter analysis of the problem domain (i.e., integration) and strict constraints on progress in order to obtain a relatively straightforward solution.

Heuristic Search

In "The Search for Generality" [45], Newell finds that the most frequent organizational structure used in Artificial Intelligence programs is one he calls heuristic search. We shall call programs which employ this organization as the sole or central organizational device HS programs. SAINT is an example of an HS program. HS programs can be considered to be programs which attempt to generate a path from a starting node A (usually the statement of the problem to be solved, given in the internal representation) to a terminal node B (usually the last link necessary to find a solution to A). The path from A to B consists of one or more nodes which are (again, usually) in the same problem domain as A and B. Thus in a theorem proving program the nodes would represent statements of possible theorems and in SAINT the nodes represent expressions to be integrated. From each node the program is able to generate one or more successor nodes. All of these successor nodes could be examined to determine if they lead to a solution (a "B" node), but it is in the nature of AI problems that if this were to occur the

program would consume too much time and space. Hence heuristics are used to select a set (possibly a null set) of successor nodes for examination in preferance to others. The use of such heuristics leads to the "heuristic" term in "heuristic search." The process of examining nodes in the tree which is generally produced leads to the "search" term in "heuristic search."

There are many strategies for guiding the search of the tree. However several stand out and deserve to be mentioned. One strategy is called "depth first." It usually selects the last node generated as the one to be examined next. This strategy has the effect of forcing an examination of a single path until it either leads to a solution or the program decides that it will not yield a solution. Such a strategy is employed in most game playing programs. At the other extreme is a strategy called "breadth first" which selects the node which was generated earliest. Such a strategy was used in the Logic Theorist [44]. SAINT chooses the node which represents an expression which it deems to be one of the simplest subproblems to be integrated.

We wish to clarify the sense in which we refer to a program as an HS program. The fact that a subroutine in a program uses heuristic search does not always imply that the program is an HS program. For example if SAINT's simplifier had used heuristic search in order to simplify expressions, then this fact does not imply that SAINT is an HS program (for example SAINT could have been just a table look-up program). Nor is it the case that any program which performs search even if the search is guided by heuristics is always an HS program. We wish to reserve this

name to programs which rely on conducting a search in the same domain in which the problem is posed. Thus programs which search for a plan in a different space from the one in which the problem is posed and thereafter find the solution immediately are not HS programs.*

The Trend toward Generality

One of Newell's other conclusions in "The Search for Generality" is that AI programs have tended in the recent past to shy away from dealing with complex problem domains such as chess, geometry, or integration, and have increasingly concerned themselves with generality. By programs which emphasize generality we shall mean programs which are concerned with an examination of mechanisms (e.g., heuristic search) which are useful in many problem domains. By programs which emphasize expertise we shall mean programs which concentrate on a particular (complex) problem domain. Examples of the trend toward generality are the advice taking programs (e.g., Black [3], Slagle's DEDUCOM [59], and even Norton's ADEPT [47]). These programs solve toy problems which have been posed from time to time by McCarthy. One of the striking features of these programs is how little knowledge they require in order to obtain a solution. In fact Persson, in his recent thesis[49] which deals with "sequence prediction" seems to feel that placing a great deal of context dependant information in a program would be "cheating." This emphasis seems to be useful when one desires to study certain

* Our emphasis regarding the space to be searched may differ from Newell's. In fact our need to use intuitive definitions and rely on analogies and examples points out the lack of a firm theoretical foundation in computation, and in Artificial Intelligence.

problem solving mechanisms in as pure a manner as possible.

Slagle, too, desired to use SAINT as a vehicle for studying certain problem solving mechanisms such as "character-method tables" (for example, method A is probably useful when the problem is of type 1 or type 5--see Minsky [41] for a discussion of this technique) and "inherited resources" (Minsky [41]). We, on the other hand, intended no such study of specific problem solving mechanisms, but mainly desired a powerful integration program which behaved closely to our conception of expert human integrators (it should be noted that Slagle compared the behavior of SAINT to that of college freshman calculus students). Nonetheless our experiment with SIN may be used to modify or improve general problem solving mechanisms.

SIN, we hope, signals a return to an examination of complex problem domains. Greenblatt's chess program [22] is another example of a recent program which deals with a complex problem domain which has been considerably neglected in the last few years.

The Emphasis on Analysis

Our emphasis in SIN is on the analysis of the problem domain. This analysis is both an analysis that we performed and built into the program, but more importantly an analysis which the program makes while it is solving a problem. In order to achieve high performance in symbolic integration we did not require that the program make a very complex analysis of the situation. Nonetheless the analysis that SIN does make markedly affects the performance of the program. When SIN is solving one of SAINT's difficult problems the most noticeable difference between its performance and SAINT's is not in the increased efficiency of the

solution,* but in how quickly SIN usually manages to decide which plan to follow and the straightforward manner with which it obtains the solution thereafter.

As we shall see in Chapter 4 SIN's methods are quite similar to those used by SAINT. However SAINT does not commit itself to a particular method, but will frequently explore several paths to a solution until it finds some path which succeeds in obtaining the answer. Heuristic search is used to find this solution path. Frequently such uncertainty is necessary in SAINT because it lacks the powerful machinery that SIN possesses and relies on (e.g., the rational function package of MATHLAB). Thus SAINT is forced to search until it finds a path which leads to subproblems that it can solve. For example, in $\int \cot^4 x \, dx$ SAINT cannot obtain a solution by using the substitution $y = \tan x$ which leads to $\int \frac{1}{y^4(1 + y^2)} \, dy$ since it cannot integrate the rational function. Thus SAINT is forced to contain a further substitution $y = \cot x$ which SIN can easily afford to ignore. In other cases the large number of subproblems proposed by SAINT arises when SAINT employs methods which do not perform a sufficient analysis or possess sufficiently tight progress constraints. For example in $\int \frac{x^2 + x}{\sqrt{x}} \, dx$, SAINT will consider transforming the quadratic in the numerator, though this transformation is not reasonable when one considers the square-root in the denominator In this problem SIN would note the square-root and would make a substi-

---

* Though SIN solves SAINT's problems about two orders of magnitude faster than SAINT's published figures, this statistic is deceptive. If SAINT were to be run under optimum conditions, SIN would only be about three times as fast on the average. The principal reason for this fact is that most of the processing time in SIN is spent in algebraic manipulation (e.g., simplification), and the cost for these operations is fairly constant in SIN and SAINT (see Appendix C).

tution which would rationalize the denominator.

We feel that SAINT is not the only HS program in which greater analysis would yield improved results. In the MATER program of Simon and Baylor [2], heuristic search is used to find a mating combination in chess. When MATER considers the set of replies that Black might be able to make in response to a given move of White, it stores these replies in a "try list." The try list is ordered so that moves which have fewest responses are considered first. The set of moves which have the same number of replies are normally considered in a first-in, first-out manner ([2], p. 435). This leads to a breadth-first search. Had the moves been stored in a last-in, first-out manner a depth-first search would have resulted. This search would mean that the program would explore a path until it became worse than some other path in contrast to MATER's criterion that a path is abandoned when it is no better than some other path. This slight change in the strategy of the program would lead MATER to find solutions to some problems on which it ran out of space, and would not materially affect its performance otherwise. This analysis of MATER is due to Henneman [26].

While we do not wish to suggest that a radically improved performance can be had in all HS programs through greater analysis, we certainly want to emphasize the effect that such analysis can have on many HS programs. Since any nontrivial analysis requires a good deal of context dependent information, we also wish to emphasize the need for such information in problem solving programs. In the long run, of course, complex analyses and strategies will have to be represented in

specialized languages. We would like to see this development occur in the Greenblatt program, for example.

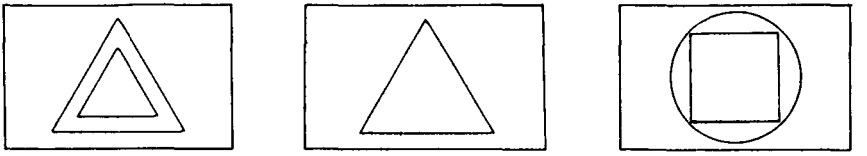
The Three Stages of SIN

SIN is a three stage program. In this respect already the organization of SIN differs from most AI programs which are composed of a single stage with a heuristic search as its principal organization. The multiplicity of stages allows the programs to devote increasing effort in later stages.

Stage 1 of SIN uses a method (Derivative-divides) which solves most commonly occurring problems. The experiment in Appendix C indicates that this method solves half the problems attempted by SAINT. Some problems integrated by this method are: $\cos x$, xe^{x^2}, $\tan x \sec^2 x$, $x\sqrt{1+x^2}$.

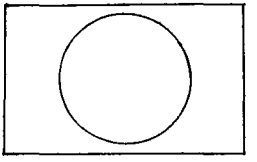
We feel that all too few AI programs employ the fact that in many problem domains there exist methods which solve a large number of problems quickly. SAINT did employ this idea in its IMSLN (IMmediate SoLutioN) routine (see Chapter 4). However IMSLN is not as powerful as SIN's first stage. Evans' ANALOGY program [17] which is one of the few AI programs which does not rely on heuristic search also could have profited from a first stage method. Evans' program deals with geometry analogies. Instructions given to humans taking a test based on these analogies are as follows: "Find the rule by which figure A has been changed to make figure B. Apply the rule to Figure C. Select the resulting figure from figures 1-5." Evans' program performs as if it were following the instructions: "Find the rule by which figure A has been changed to make figure B. Also find rules which transform figure C to each of the figures 1-5. Select the answer figure which corresponds to a transformation

which most closely fits a transformation from A to B." The test makers are essentially suggesting that one should guess the answer figure. This scheme, we have found, is effective in almost all the problems attempted by ANALOGY. Consider the figures A, B, C below:



A B C

A reasonable guess of the answer using the test makers' advice is:



TRIAL ANSWER

If such a figure is present among the answer figures then one should choose that answer. All that would be required for this step is that one test the guess for an identity with the answer figures. If this scheme should fail to find an answer, then one would enter a second stage in the program in which one would "debug" the previous guess or employ an analysis similar to Evans'. Yet once one is forced to enter a second stage, one has a piece of information that one did not previously possess--that the problem is relatively difficult. Such information may be used to guide further processing. A further use of guessing will be indicated below in discussing the Edge heuristic.

The second stage of SIN is the stage in which we spent most of the programming effort. In this stage the program is able to apply eleven highly specific methods. The principle feature of this stage is that

the program decides which method, if any, is applicable to a problem quite quickly. We shall call the manner by which this stage of SIN operates hypothesis formation. The routine at the heart of the hypothesis formation mechanism in SIN is called FORM. FORM checks for local clues in the integrand in order to generate an hypothesis regarding which method is likely to be applicable. Currently FORM can decide on the applicability of all but three of the eleven methods by using local clues. For example, if FORM notes the subexpression sin(x), then FORM will call the method which handles trigonometric functions. The first step that any of the methods in this stage is supposed to make is to verify the hypothesis that it is able to perform a transformation which will either solve the problem or simplify it. Thus if the routine which handles trigonometric functions does not believe that it is applicable to the problem, as in $\int \sin x\, e^x dx$, then it will return the value FALSE to FORM. In that case FORM might entertain a second hypothesis. Otherwise the method will continue to work on the problem.

More generally we think of hypothesis formation as a three step process. First one analyzes the problem in order to obtain an hypothesis regarding the solution method. Then the hypothesis is verified by the method prior to attempting a solution of any subproblems. Finally, if the method appears applicable then it is used in an attempt to solve the problem. If the method does not appear applicable, a new hypothesis may be generated.

We think of hypothesis formation as a model for a planning mechanism. As with any planning device one should strive to incorporate into the planner a great deal of knowledge regarding the capabilities of the rest

of the program. One aspect of the understanding that FORM has of SIN's routines is incorporated in its ability to "make the problem fit the method." By this phrase we mean that FORM is able to eliminate certain ambiguities in the problem. These ambiguities arise when certain subexpressions in the statement of the problem hinder the recognition of the true nature of the problem. For example, the analysis that FORM makes of a problem allows it to suspect that an expression is a quadratic in x even though SCHATCHEN (see Chapter 3) did not match the expression to a quadratic. This occurs when FORM is examining a square-root of a rational function. Let us suppose that none of the methods that FORM has available in this case decide that they are applicable. FORM will now attempt a further analysis because such a subexpression usually represents a block to a solution. FORM considers two excuses for the fact that the methods did not seem to be applicable. Both relate to SCHATCHEN's matching capabilities. The first is that the rational function inside the square-root was not expanded (e.g., $x(1 + x)$); the second that the rational function was not completely rationalized (e.g., $x + \frac{1}{x}$). FORM will therefore determine if these two transformations are applicable to the rational function. If they are, it will reanalyze the problem to determine if its methods are applicable. Thus FORM's analysis enables it to localize the difficulties in a problem, and its understanding of the rest of SIN allows it to find excuses for certain events and helps it to overcome the difficulties in a problem. In some of the cases just considered SAINT would have performed the same transformation (only expansion, though). Yet these transformations would be applied to the whole integrand and not to selected portions of it.

The third stage of SIN is the place that we reserved for general methods of integration. Such methods either search a great deal or involve much analysis and machinery. Hence we feel that they should be considered as a last resort. The experiment described in Appendix C indicates that only two problems required a method in this stage. The most interesting method of stage 3 is Edge which is based on the Edge heuristic and is discussed in Chapter 5. Edge is a novel integration method since it guesses the general form of the integral. Once a guess has been made, a "differencing" technique similar to GPS's [43] is applied to obtain the answer. As will be seen in Chapter 5 the guess is closely related to the antiderivative of a selected subexpression in the integrand.

CHAPTER 3

SCHATCHEN - A MATCHING PROGRAM FOR ALGEBRAIC EXPRESSIONS

Introduction

Our aim in this chapter is to develop a set of requirements for a language in which one can describe concisely and precisely algorithms for the manipulation of algebraic expressions. Several attempts at such languages have been made in the past. We would like to distinguish among these attempts two distinct approaches to an algebraic manipulation language. One could be called the command-oriented language. An example of a command would be "Let w be the name of the expression which results from substituting the expression named x for that named y in expression named z." It is customary to abbreviate this to something like "w = subst(x, y, z)."

The second approach can be called the pattern-directed (or production) approach. An example of a statement in such a language would be "x+x → 2*x," which means that if the expression currently being examined matches (i.e., is of the form) x+x, then it is replaced by the expression 2*x. Such statements will be henceforth called rules. A rule is composed of two parts, a pattern-match part (antecedent) and a replacement part (consequent).

A command-oriented language is desirable for man-machine interaction because the human is able to perform the desired pattern recognition by himself most of the time (see Martin [37], Engelman [15]). It is also useful in those situations in which the algorithms being coded are straight-forward, that is, nothing unusual is likely to happen. An example of such a situation is a program which solves a system of linear equations with variable coefficients (see ALPAK [6]).

When the algorithms being coded become increasingly complex, the pattern recognition requirements of the algebraic manipulation language are increased. To meet these requirements, highly command-oriented languages, such as FORMAC [5], include some pattern recognition facilities (e.g., the PART command). However, these facilities are woefully inadequate for many purposes (e.g., simplification, integration) and the need for a pattern-directed subset of an algebraic manipulation language has become clearly established.

In this chapter we shall be concerned solely with the pattern-directed approach. At first, we shall rely principally on the reader's intuition and understanding of algebraic expressions. Our discussion will become more and more precise as we proceed.

We shall first examine the requirements of the pattern-match. The requirements of the replacement part, which are simpler, are examined later. An application to simplification of the SCHATCHEN program which fulfills these requirements will then be discussed. The

chapter ends with an essay on simplification.

Below "PLUS", "TIMES" will designate the usual arithmetic operations of addition and multiplication. The former will also be designated by "+", and the latter by concatenation. "EXPT" will represent exponentiation.

The Pattern-Match

Let us consider the intuitive pattern for a quadratic in x -- namely, pattern P1:

(P1) $Ax^2 + Bx + C$

All would grant that the expression E1 satisfies the pattern P1 with the values for

(E1) $3x^2 + 2x + 5$

A, B, C, being 3, 2, 5, respectively. Such an expression also appears to offer no difficulties to a matching program since there is a 1 - 1 correspondence between the elements in the expression and the elements in the pattern. Thus, a straight-forward left-to-right scan should yield the corresponding values for A, B, C and result in a match. Consider, however, the expression E2. E2 is also a quadratic in x. Yet it fails to have one of the properties that E1 enjoyed. A left-to-right scan of E2 will yield the

(E2) $3x^2 + 2x$

value 3 for A and 2 for B. However, we will have difficulty in

assigning a value to C since no term in the expression corresponds to the C term in the pattern. Obviously C should be matched with 0. We generalize the example to conclude that terms in a sum in the pattern which are missing in the expression are to be matched with 0. Likewise, factors in a product in the pattern which are missing in the expression are to be matched with 1. We should note though that extra arguments in the expression might lead to failure as in expression E3:

(E3) $4x^3 + 3x^2 + 2x + 5$

Expression E4 presents us with a degenerate instance of pattern P1. Note that the operators PLUS and TIMES which are explicitly present in P1

(E4) x^2

are missing in E4. We can introduce these operators by rewriting E4 as E4'.

(E4') $1 \cdot x^2 + 0$

Let us proceed now with matching P1 and E4'. The value 1 for A is easily obtained. The 0 term in E4' will match Bx and will result in B=0. (This process will be clarified below.) Finally, due to the requirement stated above regarding missing terms in a sum, C will be matched with 0. Then in order to match P1 with E3 we required that the match must recognize missing or implicit operators.

Let us consider how the match might determine that Bx=0 implies that B=0. In P1 we implicitly introduced the convention that constants such as x are represented by lower case Roman letters and variables such as A, B, C, are represented by upper case Roman letters. Constants must match themselves. The values of variables are determined by the pattern-match and depend on the expression. Furthermore, our knowledge of multiplication indicates that if a product involves a 0 factor, then its value is 0. (We shall ignore cases with infinite factors.) Thus, if a product is matched with 0, it is required for a factor to match 0. If Bx is matched with 0, then since x must match itself, B must match 0, otherwise the match fails. A complementary requirement we shall impose is that if a product is matched with 1, then each factor must match 1. This requirement is redundant since it follows from our requirement for missing arguments in a product.

In the above we have built into the match an understanding of the arithmetic laws involving 0 and 1 in sums and products. Note though that the match assumes that the expression has been simplified to some extent. Thus, the pattern Ax^2 will not match the expression $x^{4(1/2)}$ since the constant expression x^2 is assumed to match only itself.

However, information about 0-1 laws are insufficient as can be seen when we consider expression E5:

(E5) x

In some cases such an expression could pass for a quadratic. In other cases (for example, in applying the quadratic formula) such an expression is not admissible as a quadratic. Note that the match as described above will result in the value 0 for A, 1 for B, and 0 for C for expression E5. We need to be able to describe to the match that the value 0 for A is proscribed. In fact, we would like a more general facility allowing one to delimit the range of values that the variables in the match may have. We shall require that the variable must be allowed to satisfy a predicate. We shall indicate such a facility with a slash (/) as in pattern P2. In P2 we require A to satisfy the predicate NONZERO:

(P2) $A_{/NONZERO}\ x^2 + Bx + C$

In examining expression E6 we see that we will need more predicates to limit the values of A, B, C, since E6 is certainly not a quadratic in x:

(E6) $x^2 + \sin(x)\ x + 1$

Let us consider pattern P3 which takes care of the difficulty in E6.

(P3) $A_{/NONZERO\text{-}AND\text{-}NUMBER}\ x^{2+B}_{/NUMBER}\ ^{x+C}_{/NUMBER}$

Pattern P3, however, may be a too restrictive condition. It requires

that A, B, C, be numbers.

For example, P3 will reject expressions E7 and E8

(E7) $x^2 + \pi x$

(E8) $x^2 + x + y$

since π does not appear like a number and since y is certainly not a number. If we wish to accept both E7 and E8, pattern P4 might be suitable:

(P4) $^{A}/_{\text{NONZERO-AND-FREEOFX}}\ x^{2+B}/_{\text{FREEOFX}}\ x+C/_{\text{FREEOFX}}$

We shall assume that the predicate FREEOFX determines whether an expression contains an occurrence of x and has the value T (true) if it does not contain such an occurrence.

We thus can see that the predicate facility is both a blessing and a headache since it forces one to consider quite carefully what it is that he desires to be matched.

Further complications arise when we consider the expression E9. We recognize E9 to be a quadratic.

(E9) $x + x^2$

However, in doing so we made use of the fact that addition was a commutative operation. This leads us to require that the match must take into account the commutativity of addition and multiplication. (Non-commutative addition and multiplication could be represented with different operators than PLUS and TIMES.) As it turns out this

requirement increases the cost of the match greatly. It is now insufficient to perform a single left-to-right scan of the expression. We may be forced to traverse the expression several times. We shall assume, however, that the pattern is to be scanned once from left-to-right. This will allow us to use the values of previously bound variables. For example, a pattern for determining whether an expression is a perfect square might be written as P5

(P5) $A_{/NONZERO-AND-FREEOFX}\ x^2+B_{/FREEOFX}\ x+C_{/FREEOFX}$ -

AND - $(B^2-4AC = 0)$

since by the time we encounter C, the values for A and B should already be known or else the match has already failed.

The predicate facility is one way in which the pattern can be used to direct the match. Below we shall give descriptions of other facilities and examples in which they might be used. These facilities are made available by the use of modes for the variables in the match. The desirability of the first of these modes is indicated in expression E10.

(E10) $3x^2y + 2x + 1$

The difficulty in matching expression E10 is due to the occurrence of more than one factor (other than x^2) in the terms involving x^2. We would really be interested in having the variables A and B act as coefficients of x^2 and x, respectively. This means that

in the term involving x^2, the product of all the other factors is a candidate for A. To show this we shall use the indicator COEFFT (coefficient in TIMES) as a modifier for A as is shown in P6:

(P6) ^{A}/COEFFT,NONZERO-AND-FREEOFX x^{2+B}/COEFFT,FREEOFX ^{x}

$^{+C}$/COEFFP,FREEOFX

In P6 we used the indicator COEFFP (coefficient in PLUS) to modify C. This means that C will match the sum of the remaining terms in the expressions. The result of matching P6 with E10 is : A=3y, B=2, C=1.

In expression E11 we see another phenomenon which will necessitate the addition of a new mode. In E11

(E11) $$2x^2 + \sqrt{2}x^2 + 3$$

there occur two terms involving x^2. If we assume that each term in the pattern should match exactly one term in the expression, then the single term Ax^2 in the pattern will fail to account for the two terms in E10. We need a facility for specifying to the match that a particular variable in the pattern is to be considered a coefficient in both a product and a sum. This is done in pattern P7 by using the indicator COEFFPT (coefficient in PLUS and TIMES) to modify A and B.

(P7) ^{A}/COEFFPT,NONZERO-AND-FREEOFX x^{2+B}/COEFFPT,FREEOFX $^{x+C}$/COEFFP,FREEOF

With the machinery we have developed we can now match pattern P7 with

the expression E12:

(E12) $y^3 + 3\pi x^2 y + 6x^2 + 5y + 1$

The result of this match should be $A=3\pi y + 6$, $B=0$, $C=y^3 +5y +1$.

In the above examples we were attempting to determine whether the expression was a quadratic in x. Suppose we wanted to generalize the problem in order to determine whether the expression was a quadratic in some atom, but where the atom was not fixed, but may itself change. More precisely, we desire a function QUADRATIC of two arguments EXP and ARG. This function is expected to determine whether EXP was a quadratic in ARG. P8 can be used as a pattern in QUADRATIC.

(P8) $$A_{/COEFFPT,NONZERO-AND-FREEOFARG}\left(VAR_{/EQUALARG}\right)^2 + B_{/COEFFPT,FREEOFARG}\left(VAR_{/EQUALARG}\right) + C_{/COEFFP,FREEOFARG}$$

In P8 we introduced the predicate FREEOFARG which has the obvious related function to FREEOFX in pattern P7. The predicate EQUALARG tests the value that the match assigned to VAR for equality to ARG.

Let us now consider the problem of extracting a perfect square from a sum. More precisely let us consider the situation in which a sum has three terms which are individually of the form $A*VAR^2$, $B*VAR$ and C, and whose relation is defined by $B^2-4AC=0$. This differs from

the situation described in pattern P5 in that the expression may now have more than three terms and in that the value of VAR is originally unknown and depends on the expression being matched. Our first attempt is to describe this situation with P9:

(P9) $^{A}/_{\text{NONZERO-AND-NUMBER}}\ ^{\text{VAR}^2+B}/_{\text{NUMBER}}\ ^{\text{VAR}+C}/_{\text{NUMBER-AND-}(B^2-4AC=0)}$

$$+^{D}/_{\text{COEFFP}}$$

It turns out that pattern P9 does not satisfy our requirements because there is some ambiguity regarding VAR. In predicate P8, VAR was determined uniquely by the predicate EQUALARG. In the current situation no such a priori predicate exists. The first value of VAR can be essentially anything. To indicate this we can write $^{\text{VAR}}/_{\text{TRUE}}$ instead of VAR, where TRUE is a predicate which is true on any input. However, the second occurrence of VAR in the pattern (i.e., in $^{B}/_{\text{NUMBER}}\ ^{\text{VAR}}$) is intended to be fixed. That occurrence of VAR must be the same as the previous value attached to VAR. To make this point clear, let us consider expression E13:

(E13) $\quad y^2 + 2x + 1 + 5z + 2y$

This expression will match pattern P9 with A=1, B=2, C=1, D=5z+2y, and with the first value of VAR equal to y and the second equal to x. To avoid this situation we could write the second occurrence of VAR as $^{\text{VAR1}}/_{\text{EQUALVAR}}$. This is a fairly clumsy mechanism (even though a similar device was used in P8). What we shall do instead is to

define a new mode called CONV in which the first occurrence of the variable (e.g., VAR) will satisfy the predicate (e.g., TRUE) and the latter occurrences must match the expression matched during the first occurrence. We thus arrive at pattern P10. (The CONV mode is directly related to the PAV (pattern variable) mode of CONVERT [23].)

(P10) $A_{/NONZERO\text{-}AND\text{-}NUMBER}\left(VAR_{/CONV,TRUE}\right)^2 + B_{/NUMBER}\ VAR +$

$C_{/NUMBER\text{-}AND\text{-}(B^2-4AC=0)}\ {}^{+D}/COEFFP$

Pattern P10 will match E13 with A=1, B=2, C=1, D=2x+5z, and VAR=y.

Let us consider P10 with expression E14:

(E14) $y + y^2 + x^2 + 2x + 1$

The first attempt will be to match VAR with y. This attempt will fail and the match will fail even though a perfect square exists if VAR were to match x. What is required here is a facility for directing the match to search for further possibilities. It is assumed, of course, that the user of such a facility is aware that it may cause a profound increase in the cost of a match. We shall introduce such a facility with a mode which indicates a loop over the expression. Such a facility may be used when there exists a set of variables (such as A, B, C) in pattern P10 which are mutually interrelated (e.g., $B^2-4AC=0$). This facility will direct the match to continue making trial guesses for the variables until one set is found which is satisfied or until all possibilities have been exhausted.

In programming terms the loop facility in the problem of pattern P10 will ask for a 3-level loop in which all possible values for A, B, C (note that VAR is determined along with A) are examined until one set is found which satisfies $B^2-4AC=0$. The syntax for the loop facility is given in pattern P11:

(P11) $^{A}/_{\text{LOOP(A,B,C),NONZERO-AND-NUMBER}}\left(^{\text{VAR}}/_{\text{CONV,TRUE}}\right)^2 +$

$$^{B}/_{\text{NUMBER}}\ ^{\text{VAR+C}}/_{\text{NUMBER-AND-}(B^2-4AC=0)}\ ^{+D}/_{\text{COEFFP}}$$

Although in the above we have concentrated entirely on describing patterns for quadratics, our intention has been to describe a set of requirements for a language which can handle a far richer set of tasks. To indicate the power of the machinery we have developed, we shall give below a pattern which tests for the occurrence of $\sin^2 B + \cos^2 B$ in a sum. Pattern P12 will match expression E15 and results $A=5\cos^2(y) + 1$, $B=2x$, $C=2$, and $D=3y+2\sin^2(x)$.

(P12) $^{A}/_{\text{COEFFPT,LOOP(A,C)}},\ ^{\text{NONZERO}}\sin^2\left(^{B}/_{\text{CONV,TRUE}}\right) +$

$$^{C}/_{\text{COEFFPT,NONZERO}}\ \cos^2(B) + {}^{D}/_{\text{COEFFP}}$$

(E15) $3y + 2\sin^2(x) + 5\sin^2(2x)\cos^2(y) + 2\cos^2(2x) + \sin^2(2x)$

The implicit relationship between A and C in pattern P12 appears fairly trivial -- that is, both A and C must be nonzero.

However, expression E15 shows that the loop facility helps to get us out of the trap of matching B to x in the $2\sin^2(x)$ term.

We have so far neglected a discussion of the matching requirements of patterns which include exponentiation. We have let intuition guide us through the cases where exponentiation did occur in the patterns above. As before a constant expression in the pattern of the form A^B (e.g., $\sin^2(x)$) must match itself. Otherwise, if A^B is to be matched against the expression 0, we shall assume that it is necessary and sufficient for A to match 0. (The difficulty that arises if B likewise were to match 0 is ignored.)

If A^B is matched against 1, then either B must match 0 or A must match 1. Note that this can lead to a difficulty if both A and B are variables, since only one value will be determined. If A^B is matched against $E_1^{E_2}$, then B must match E_2 and A must match E_1 or B must match 1 and A must match $E_1^{E_2}$.

In pattern P13 we are testing for an expression of the form $\sin^n(x)\ \cos^m(x)$. This pattern will match the expression $\sin(x)$ and result in the values N=1, M=0.

(P13) $\sin^{N/\text{INTEGER}}(x)\ \cos^{M/\text{INTEGER}}(x)$

Pattern P14 is included here to indicate some of the ambiguity that is inherent in patterns.

(P14) $\left(A/\text{NONZERO-AND-FREEOFX}\ x^{N/\text{INTEGER}} + B/\text{FREEOFX}\right)^{M/\text{INTEGER}}$

P14 corresponds to the intuitive pattern $(ax^n+b)^m$. When P14 is matched against $(x^2+1)^3$ it will yield A=1, B=1, N=2, M=3. When it is matched against x^6 it will yield A=1, B=0, N=1, M=6, although A=1, B=0, N=2, M=3 serves equally well as a set of solutions. We used this pattern to indicate some of the limitations of the matching program we have been defining. In the case of the expression x^6, we obtain via pattern P14 the implicit relation NM=6. This means that we have given the program insufficient information regarding the choice of values for N and M in this case. The match cannot be expected to do very well in this instance.

A second difficulty with pattern P14 which has already been mentioned occurs when it is matched against 1. In this case our requirements for the match indicate that all that shall result is M=0. We could have obtained A=0, B=1 if the requirements regarding the matching of 1 had been reversed. Neither situation is wholly satisfactory. However, it is hard to foresee a compromise solution which will be wholly satisfactory.

The lesson that is learned from pattern P14 is that it is up to the user to make his patterns sufficiently restrictive so as not to yield ambiguous situations in those cases in which they are likely to be applied.

The impression that is likely to be in the minds of some readers is that more machinery is yet to be described. We do not intend to do this. In some strong sense the design of a good algebraic

manipulation language is never over. Any description is only an imperfect solution to many conflicting requirements. What makes a language interesting is its usefulness in solving problems. The set of requirements described above should satisfy this criterion for many problems.

Before we end our discussion of the match and turn our attention to the replacement part of the rule, there are a few remarks which are in order.

The match that we have described is based on the form of the expression. Frequently, we desire to know information regarding the form to which the expression could be reduced under legal algebraic transformations. When we ask "Is this expression a quadratic in x?" we usually mean "Is this expression equivalent to a quadratic in x?" rather than "Does it look like Ax^2+Bx+C?". Thus expressions E16 and E17 are quadratics in x which do not look like quadratics in x unless we stretch our imagination a good deal. By restricting ourselves to a match based on form we can hardly expect this match to determine that E16 and E17 are quadratics.

(E16) $x^2 + \sin^2(x) + \cos^2(x)$

(E17) $(x+1)\ (x+2)$

The generality of the match means that its power is restricted. One could, of course, design a special-purpose test for a quadratic in x. It might check to see if the third derivative of the expression with

respect to x is equivalent to 0 and if the second derivative is different from 0. Theoretical results by Richardson (see Appendix B) indicate that there will be problems even with such a special purpose match which it could not determine correctly in finite time. Special purpose devices probably could be designed for each pattern that could be written for our match. Some of these would have to be quite ingenious in order to be more powerful than our match. These devices might be necessary in certain situations. However, they run counter to our desire for a language in which one can write concise rules.

We shall have more to say about the pattern match when we discuss the existing algebraic manipulation languages below.

Replacement

Having discussed the matching part, we shall now describe the process by which new expressions may be generated using the results of the match. This process we shall call the replacement part of the rule.

Let us consider the intuitive statement of rule R1:

(R1) $$Ax^2 + Bx + C \rightarrow Ay^2 + C - \frac{B^2}{4A}$$

A successful match of the left-hand-side of R1 should result in a dictionary containing the values of A, B and C. This dictionary

is then used to generate the right-hand-side expressions by replacing the variable names by the values which were assigned to them during the match. If we consider the expression x^2+2x+1, the match should result in A=1, B=2, C=1 and the rule should yield the expression $1y^2+1-\frac{2^2}{4\cdot 1}$. Since this expression is unsightly we shall require that the replacement step should simplify the expression. Thus, R1 would result in the expression y^2. (Note that R1 performs the operation of completing a square.)

Suppose we were given rule R2:

(R2) $\cos(nx) \rightarrow \cos^n(x) - \binom{n}{2}\cos^{n-2}(x)\sin^2(x) + \binom{n}{4}\cos^{n-4}(x)\sin^4(x)$

R2 computes the first 3 terms in the expansion of cos(nx) in terms of cosx and sinx. If we had matched the expression cos(4x) with rule R1, we would result in an expression involving the combinatorial terms $\binom{4}{2}$ and $\binom{4}{4}$. In order to have an expression amenable to further computation $\binom{4}{2}$ and $\binom{4}{4}$ should be evaluated to yield 6 and 1, respectively. Thus, we require a facility for evaluating selected portions of the expression. With this facility R2 can be written as R3.

(R3) $\cos(nx) \rightarrow \cos^n(x) - \text{EVAL}(\binom{n}{2})\cos^{(n-2)}(x)\sin^2(x) + \text{EVAL}(\binom{n}{4})\cos^{(n-4)}\sin^4(x)$

The replacement routine will substitute for each atom which appears in the right-hand-side, its value in the dictionary if there is such a value. If no such value exists, the atom will be replaced by itself, that is, it will be quoted. We will require a supplementary quoting mechanism so that we may use right-hand-sides in which names of variables appear which are not replaced. An example of a rule using such a facility is R4. DIFF(A,B) is assumed to yield the formal derivative of A with respect to B.

$$(R4) \qquad f(x)^{g(y)} \rightarrow f(x)^{g(y)} \quad \text{EVAL (DIFF(g(y),(QUOTE x)))}$$

Although for expository purposes we used only intuitively written pattern matches in the rules above, it should be clear that in practical situations the left-hand-sides of the rules would be replaced by more explicit matching forms.

Existing pattern-directed languages

The requirements given above for a matching and a replacement program are satisfied by the SCHATCHEN* and REPLACE routines used in SIN. We would like to place these programs in their historical context. SCHATCHEN has been most influenced by ELINST (ELementary INSTance), a set of routines included in Slagle's SAINT for the purpose of matching algebraic expressions to forms. ELINST satisfies many of the algebraic properties of SCHATCHEN such as variable arguments to PLUS and TIMES, missing operators, and commutative operators. It differs in that it does not give the user explicit control mechanisms of the scan of the expression. ELINST will generate all possible sets of values for the variable and only then will it apply the side relations to determine those which satisfy the pattern. Besides this weakness, ELINST suffers most by being essentially undescribed. I suspect that had Slagle described ELINST in 1961, then some of the proposals for algebraic manipulation languages which were made since 1961 would have had a different character. ELINST had to be as general as it is because the problem that Slagle was trying to solve required such generality. Furthermore Slagle encountered grave problems in fitting his program into the memory (32K) of the 7094 and thus chose to make use of the economy of calls to ELINST in many situations in which it would otherwise have been wiser to write special purpose matches. Thus he claimed that one half of the time that was spent usefully by SAINT (i.e., excluding

*match-maker in Yiddish

garbage collections) was spent in pattern recognition.

The features of the algebra-oriented pattern-directed languages that were introduced in the past six years (e.g., AMBIT [10], FORMULA ALGOL [48], Fenichel's FAMOUS [19], PANON-IB [8])* appear to have a great deal in common. PLUS and TIMES are restricted to at most two arguments. Operators that appear in the pattern must explicitly appear in the expression. Sometimes also PLUS and TIMES are not recognized as commutative operators. All these restrictions mean that the patterns are highly specific and that several rules are necessary in order to accomplish a task that can intuitively be stated in a single rule. The advantage that such matching routines have over a more general one such as SCHATCHEN is that each of the rules is quite readable and relatively efficient to execute. However the effect of a set of rules which is equivalent to a single SCHATCHEN rule is probably harder to guage than the SCHATCHEN rule itself. The execution time of a set of rules is also probably longer than the execution time of a single SCHATCHEN rule.

Here is the kind of rule set that would be required in such languages in order to recognize a quadratic in x:

(RS)

x^2	ax^2
$x^2 + bx$	$ax^2 + bx$
$x^2 + x$	$ax^2 + x$
$x^2 + bx + c$	$ax^2 + bx + c$
$x^2 + x + c$	$ax^2 + bx + c$
$x^2 + c$	$ax^2 + c$

*It should be noted that these languages have a greater generality than a discussion of their usefulness in matching algebraic expressions would indicate.

In proposing the above twelve rules we are assuming that the language provides for commutativity in PLUS and TIMES and for the ability for declaring a, b, c to be FREEOFX. In systems in which a minus sign is recognized as a distinct operator one might require even more rules. Unfortunately the rule set proposed is not as powerful as Pattern P7 because each term in the pattern will be matched with exactly one term in the expression. It appears that one could overcome this restriction only by a recursive or iterative application of the rules. In fact, the FAMOUS system relies on the fact that the rule set is applied repeatedly to a given expression although in FAMOUS' case the reason for this reliance has a deeper philosophical significance owing to Fenichel's strong affirmation of the concept of local transformation embodied in Π-theory.

In our previous discussion we have emphasized the desirability of the implicit arithmetic operators PLUS, TIMES and EXPT in the pattern. There are, however, instances where the operator must explicitly be present. In the rule below which is used for rationalizing sums in a recent thesis by Iturriaga [28],

(R5) $\qquad A+B/C \rightarrow \frac{A\times C+B}{C}$

the "+" operator must be present as well as the "/" operator. It is possible to simulate the requirement that these operators must

be present by requiring that A cannot be 0 and that C cannot match 1. However such a situation is clumsy at best, and a facility for explicit operators should be provided. With such a facility for explicit operators (present in the early versions of SCHATCHEN, but dropped because of lack of use), a user of the algebraic manipulation system will be capable of programming in a wide variety of styles. These will range from the fairly rigid and inflexible rules of the rule set RS to the type of rule exemplified by pattern P11.

We shall also mention a slight controversy regarding the number of arithmetic operators which should be present in the internal structure of an algebraic manipulation system. Some people appear to believe that a large number of operators including unary minus, quotient, and difference is a good idea. Experience has shown, however, that such systems, expecially when combined with an inflexible pattern-match, require an increase in the user's awareness* which tends to downgrade his problem solving ability. The less a user must be concerned with what is actually happening, the more likely he is to solve hard problems. Of course, if the details which are hidden in the system involve exponential growth or the like, hiding such details can be disastrous. This is not, however,

*"Awareness" is a term used by Weizenbaum to indicate the degree of attention to detail which a user is required to maintain in a given situation.

the situation when arithmetic operators are translated internally into only three - PLUS, TIMES, and EXPT. At the input-output level, just the opposite effect takes place. Here we wish to let the user of the algebraic manipulation system have the flexibility with which he feels comfortable. The recent trend in input-output of algebraic expressions has been to have this flexibility (see Martin [37]).

Implementation of SCHATCHEN

SCHATCHEN is currently implemented as a set of LISP programs. Several people have suggested that one should embed it in a more general language. CONVERT [23] seems to be the regnant choice for such a language. CONVERT is a general pattern directed language with much machinery for the transformation of list structures. In fact, two modes in CONVERT which were introduced in the past year (i.e., BUV - bucket variable - and UNO - unordered search) were introduced by Guzman and McIntosh, the designers of CONVERT, with the intention of such embedding. Interestingly enough, the BUV mode is sufficiently general that it has replaced other CONVERT modes. The advantage of such an embedding is that it would allow the user to employ other facilities of CONVERT. These facilities are quite impressive. The major disadvantages are due to inefficiencies in a straight-forward implementation. In order to discuss these inefficiencies we will have to describe the manner in which SCHATCHEN performs a scan.

Suppose we have a pattern of form I,

(I) P1 + P2 + P3

and an expression of form II.

(II) E1 + E2 + E3 + E4

The scan proceeds by attempting to match P1 with E1. If that fails an attempt will be made with P1 and E2, then P1 with E3. If P1

matches E3, then E3 will be deleted from II, and the scan proceeds by matching P2 + P3 with E1 + E2 + E4. This deletion is done by using the RPLACD subroutine of LISP. In general this is an unsafe method. It means that any prior references to II will refer to the expression with E3 deleted, which can be disastrous. However, great care is used inside SCHATCHEN to maintain pointers to the excised expression and to restore it to its original shape once the match has been performed. Furthermore, all the pointers that a pattern can have to intermediate results are carefully copied. The alternative to the deletion approach is to completely reproduce expression II without E3. The alternative is quite costly especially when the number of failures in identification is taken into account. Suppose patterns P1 and P2 are related via a loop, then P1 may have to be rematched after an original successful match. More likely is the case that P1 is matched with E3, but P2 finds no match at all and thus the match fails. The method of reproducing an expression entirely following a match of a subpattern with a subexpression is thus seen to be quite expensive. A normal string transformation language or even a list transformation language such as CONVERT (except for the UNO mode) does not face this difficulty because the scan along both the expression and the pattern is left-to-right. Thus, if P1 matches E3, P2 can only match subexpressions to the right of E3, (i.e., E4). When one introduces commutativity into the picture, the situation becomes more complicated. Thus, in our example, after P1

matches E3, we must start P2 with E1, P2 with E2, P2 with E4. It is the commutativity requirement which necessitates the rescan of the expression.

An alternative to the SCHATCHEN scan is to scan left-to-right along the pattern with each subexpression. Thus, if E1 does not match P1, then a match is attempted between E1 and P2. With this scan one is forced to keep intermediate results and perform complex processing at the end of the scan in order to determine whether the variables of the match satisfy their predicates and are properly related. This alternative was rejected as being too unwieldy.

Another aspect of the implementation of SCHATCHEN turns out to have important semantic properties. Intermediate results in SCHATCHEN are stored in a special list called ANS. On this list we also find the excision information mentioned above as well as markers used to indicate levels of scope of variable bindings. A successful technique in using SCHATCHEN is to use predicates which are themselves calls to SCHATCHEN and which introduce new variable bindings to the ANS list. Thus, a variable A may be required to be of the form BC, where B and C must match certain patterns. By calling SCHATCHEN directly as the predicate for A, then the values of B and C would be lost. However, if one calls a routine exactly one level below SCHATCHEN (namely M1), then one can preserve the values of B and C in the final result as well as obtain the full power of SCHATCHEN

The fact that ANS is available for all to use during the match can be dangerous since the predicates could accidentally destroy a great deal of information. Nonetheless the advantage of such an implementation device far overrides this difficulty. The ANS mechanism represents another difference between CONVERT and SCHATCHEN. CONVERT does not allow direct access to its dictionary. Many of the modes in CONVERT, however, perform some change to this dictionary. In this regard it should be noted that FLIP [62], another pattern-directed language which is similar to CONVERT in emphasizing the transformation of lists, concentrates on the control of the scan by the user. FLIP, however, lacks much of the recursive machinery of CONVERT and thus appears to be less likely a candidate for a language in which to embed SCHATCHEN.

A Partial Description of SCHATCHEN

SCHATCHEN has two arguments, an expression and a pattern. These will be denoted e and p, respectively. Variables in the pattern are written in the form (VAR name pred argl ... argn) where

name = name of variable

pred = predicate associated with the variable

argi are arguments 2 through (n+1) of pred.

The first argument of pred is assumed to be the expression that the match assigns to the variable.

If a variable has a mode, the mode is written in prefix form. Thus, $^{A}/_{COEFFPT,NUMBER}{}^{x}$ becomes (COEFFPT (VAR A NUMBER) x), and $^{A}/_{COEFFP,EQUAL\ 5}$ becomes (COEFFP (VAR A EQUAL 5)). (This pattern tests for the equality of the variable A with 5.)

SCHATCHEN (e p)

If e equals p, the match succeeds.
If p is of the form (VAR name pred argl, ..., argn), then pred (e argl arg2, ..., argn) is evaluated. (Note that argl, ..., argn are replaced using ANS, SCHATCHEN's internal push down list. If they contain names of variables on ANS the most recent corresponding values are used. Otherwise, EVAL (the LISP interpreter) will obtain the value of the variables). If the value of pred is TRUE, the match succeeds and ((name · e)) is appended to ANS. Otherwise the match fails.

If p is of the form (op p1 ... pn) and op is not PLUS, TIMES or EXPT, then e must be of the form (op' e1 ... en) and each pi must match ei and op must match op'. Otherwise the match fails.

If the pattern is of the form (EXPT p1 p2), then 1) e is (EXPT e1 e2) and p1 matches e1 and p2 matches e2
or 2) e is 0 and p1 matches 0
or 3) e is 1 and a) p2 matches 0 or b) p1 matches 1
or 4) p2 matches 1 and p1 matches e
Otherwise the match fails.

If the pattern is of the form (op p1 p2, ... pn) and op = PLUS or TIMES, then if e is not of the form (op e1, ..., em), e is transformed to (op e). In this case an attempt is made to match each pi with some ej. The scan starts with p1 matched with e1. If that fails p1 is matched with e2. If pi matches some ej, ej is deleted (using RPLACD) from e and the scan continues with pi+1 matched with the first subexpression remaining in e. If for some pi no ej can be found to match it, then pi is matched with 0 if op = PLUS of 1 if op = TIMES. If that also fails, the match fails. If all the pi have been matched, but some ej have not, the match likewise fails.

Exceptions to the treatment above are due to modes. If op =PLUS, and pi is of the form (COEFFPT (VAR name pred arg1, ..., argn) p1,..., pk), then the remaining expression is traversed with the pattern (COEFFT (VAR name pred arg1, ..., argn)p1, ..., pk). Each subexpression that is thus matched is deleted from the expression. The simplified sum of the results of the scan becomes the value of the variable and is appended to ANS. If no subexpression could thus be matched, then pred(0, arg1, ..., argn) is attempted. If this too fails, the match fails.

If op = PLUS and pn is of the form (COEFFP (VAR name pred arg1,...,argn)) then if e is currently of the form (PLUS ei, ..., en), then pred (e arg1, ..., argn) is evaluated. If the value of pred is true

((name. e)) is appended to ANS. If no subexpressions remain in e then pred (0 argl, ..., argn) is attempted. If it succeeds, ((name. 0)) is appended to ANS. Else the match fails.

If op = PLUS and pi is of the form
(COEFFT (VAR name pred argl, ..., argn)pl, ..., pk), then (TIMES pl, ..., pk) is matched with e. If the match succeeds and e remains of the form (TIMES el, ..., en) then pred (e argl,...,argn) is attempted. If it fails, the match fails. If no subexpressions remained in e, then pred(1 argl, ..., argn) is attempted. If this succeeds ((name. 1) is appended to ANS. Else the match fails. All other matches fail.

An Application of SCHATCHEN

SCHVUOS - SCHATCHEN'S VERSION OF AN UNASSUMING OPERATIONAL SIMPLIFIER

Owing to space considerations of the 7094, SIN required a small but powerful simplification program. Such a program, called SCHVUOS, was written and it gained both its power and small size by capitalizing on SCHATCHEN's matching capability. SAINT's simplifier was a LAP (the machine-language assembler for LISP) coded subroutine written as a Master's thesis by Goldberg in 1959 [21].

SCHVUOS does not assume a standard (canonical) form of an expression. This means that it will be slow when the expressions to be simplified are large. In integration, however, it is rare to encounter large expressions. The speed gained by a canonical order can be seen in the following example. Suppose, two simplified expressions are to be added. If the expressions are to be canonically ordered, then the addition process is basically a merge of the expressions with a simplification occuring if two terms are identical except for a constant factor. If, however, the expressions are not ordered then we generally require a two stage process. Given a term in the second expression we must determine if there exists a term in the first expression which is identical to it except for a constant factor. This may require a complete traversal along the first expression. If the number of terms in each of the two expressions is n, this process takes on the order of n^2 term-to-

term matching steps. The canonical order scheme requires only on the order of n steps. However, some time must be spent in determining the canonical description and keeping its value around. Furthermore, the routines that generate the canonical order are usually very space consuming. Thus, the use of a canonical order is only worthwhile if the expressions are to be heavily manipulated.

As has been implied in the above, much of the program effort and execution time in a standard simplification program is spent in collecting terms in sums. Related effort is spent in collecting exponents in products. In SCHVUOS the collection of terms in a sum is handled by calling SCHATCHEN and asking it to determine the coefficient of the first term in the sum.

Suppose we had the expression E18,

(E18) $2x + 3x^2y + z + x + yx^2$

then SCHVUOS will strip the first term of the sum of its coefficient and generate the pattern P15:

(P15) $A_{/COEFFPT,NUMBER}x + B_{/COEFFP}$

SCHATCHEN will yield A=3, $B=3x^2y+z+yx^2$. Next the pattern P16 is generated on the expression B. Now SCHATCHEN will result in A=4, B=z.

(P16) $A_{/COEFFPT,NUMBER}x^2y + B_{/COEFFP}$

Note that x^2y and yx^2 are recognized as equivalent. Thus, the simplified sum is E19

(E19) $$3x + 4x^2y + z$$

The operation of collecting exponents in a product is handled similarly.

The basic simplification program requires only about two pages of LISP code in contrast to a typical LISP simplification program (such as Korsvold's[33]) which requires about 20 pages of LISP code and has the same power, for our purposes, as does SCHVUOS.

SCHVUOS contains some unusual simplification rules because of the integration environment in which it operates. Thus, arcsin(sin x) simplifies to x and sin(arccos x) becomes $\sqrt{1-x^2}$. Moreover, $e^{1+2 \log y + \log z}$ becomes y^2ze. (This transformation is also handled by a call to SCHATCHEN.)

The simplification of an expression is done recursively. Each operator (e.g., PLUS) first simplifies all its arguments. The exception is TIMES which results in 0 if any of its arguments is 0.

It is possible to achieve an economy if expressions which have been simplified in the past are not simplified redundantly. This has led to the AUTSIM-bit in FORMAC [63] and to a similar device in Martin's simplification program. In SCHVUOS one can sometimes achieve this effect by setting a flag which means that the arguments of the top level operator, PLUS, say, are already simplified although their sum, say, need not be simplified. This is done in the differentiation program used in SIN.

Attitudes Toward Simplification

There seems to be a wide range of attitudes of people in the field of algebraic manipulation regarding the role that an algebraic manipulation system should play in simplification. One view, let me call it the conservative view (held by Fenichel, for example maintains that the system should not simplify expressions until specifically told to do so. In this point of view there is to be n fixed system's simplifier and no fixed canonical order of expressio The conservative view negates the view of those whom we shall call the liberals (exemplified by the FORMAC design) who believe in a canonical order, in a fixed simplifier and in implicit simplification. One might even define a third viewpoint, a radical one, in which the system will represent expressions internally in a form quite different from their external form. Rational function progra (ALPAK [6], PM[12], and MATHLAB's rational function package [36]) adopt this approach. A radical system is prone to use the distributive law indiscriminantly and to transform trigonometric function into their exponential form in order to take advantage of the power ful simplification algorithms which are then available.

Two considerations should guide one in designing an approach to simplification within a given system. The first is the generality of the system, that is the range of problems which could be reasonably solved by it. The second is the efficiency of the syste in the solution of its problem. It appears to be an axiom that the

more general a system is, the less efficient it is. The most radical attitude toward simplification usually belongs to systems which are very powerful and fast in solving problems. We must, however, adopt a rather broad outlook regarding efficiency in order to understand what makes a liberal system more efficient than a conservative one for the problems that both can handle. It is not necessarily execution time which is being decreased, it is the burden of awareness on the part of a programmer which is decreased in a liberal system. If you can make assumptions about the simplifier then you need think much less about the problem while you are programming its solution. Yet the argument for conservatism is too strong to be neglected. It relies on the axiom that the simplest expression depends on the problem being solved. Two examples which demonstrate this point and which have previously appeared (Moses [42], Fenichel [19]) are:

$\frac{4x^7}{x^{12}+1}$ can be harder to integrate than $4x^3 \cdot \frac{x^4}{(x^4)^3+1}$.

The latter strongly suggests making the simplifying substitution $y=x^4$. The former disguises this substitution but is more likely to be a result of any standard simplifier. Likewise, $\frac{1}{1+\cos x}$ may be harder to integrate than $\csc^2 x - \frac{\cos x}{\sin^2 x}$ which is equivalent to it. The former is easier to graph, read and write. The latter is immediately integrable, whereas the former requires the substitution $y=\tan\frac{1}{2}x$.

While in the above examples one can reasonably hope to transform one expression into another, this is not true of the example below. This example is intended to show that even the most obvious simplification rules can be harmful in some situations. Suppose a user generates three terms of an infinite series. We shall assume that he is attempting to obtain a general term. Suppose that the first term is 1, the second $2x+1$ and the third $3x^2+3x+1$. I maintain that if these terms were presented as $x+1-x$, $x^2+2x+1-x^2$, $x^3+3x^2+3x+1-x^3$, then the result would contain more information than before, for it would lead to a reasonable hypothesis that the general term is $(x+1)^n-x^n$. Yet one of the first rules of any existing simplifier is $x-x \rightarrow 0$.

One argument that can be given against the radical approach is given in the problem of integrating $(x+1)^{1000}$. If one expands this expression, as a rational function package is likely to do, then one will use a great deal of space and time and result in an unsightly expression. However, the expression can be easily integrated to yield $\frac{1}{1001}(x+1)^{1001}$ by leaving it in its original form. Recent information indicates that future ALPAK systems will leave expressions in their factored form in order to resolve difficulties created by problems such as this.

What then is the attitude that one should adopt toward simplifications? A reasonable one would be to use each of these attitudes where they are most useful. In cases where there is a need for a

great deal of rational function manipulation and relatively little pattern recognition one should adopt a radical attitude. When the problem is not easily framed as a rational function problem or where the computational effort is light, but where the pattern recognition is not crucial, then you adopt a liberal attitude. Finally, when a standard simplifier will lead you into difficulty you just must restrict its effect.

Separating the radical attitude within a program from the liberal one is usually easy -- there is a separate program to handle rational functions. Between the liberal and conservative modes there are too many intermediate steps. Here what appears to be required is a black-box simplifier with many inputs or indicators. With these inputs one could control the effect of the simplifier. It would be interesting to see if one could formulate a language in which a program (or a user) could communicate with the simplifier. For example, it could check certain indicators before attempting any given simplification. The cost for such checking could be quite minimal.

An example of the use of such a simplifier is represented as follows: A common simplification rule is $(ab)^m \rightarrow a^m b^m$. However, in general this rule is inaccurate (e.g., when a=-1, b=-1, $m=\frac{1}{2}$, the left-hand-side yields 1, the right-hand-side, -1, assuming a standard interpretation of the square root). If one suspects that this rule will lead to difficulty then one can leave a test condition in the

indicator for this rule which will weed out those cases in which the result is erroneous.

CHAPTER 4

SIN - THE SYMBOLIC INTEGRATOR

Introduction

In this chapter we describe the operation of SIN. At first SIN's flow of control is analyzed. Then each of the methods used is described in detail. Finally, the performance of SIN on two examples is shown. Throughout this chapter the contrast between SIN's and SAINT's approach and methods will be made clear.

Flow of Control and Subproblems in SIN and SAINT

A problem given to SIN may be said to pass through the three stages of Figure 1.

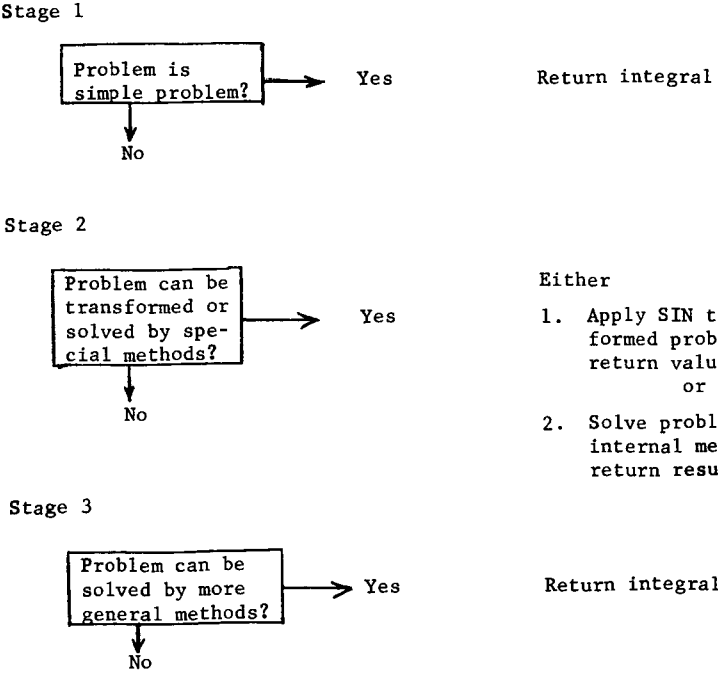


Figure 1 - The 3 Stages of SIN

As figure 1 indicates, the first stage solves simple integration problems. In the second stage, we determine whether one of about ten specialized methods is applicable to the problem. This determination is made by a routine called FORM and is quite fast. If a method is found to be applicable the problem will be either transformed and SIN will be asked to integrate the transformed problem, or the problem will be integrated using techniques internal to the methods. If no method is found which is applicable, a more general method will be called in stage 3 in order to solve the problem. In this chapter we shall describe a third stage consisting of a simple Integration-by-parts routine. In Chapter 5 we shall describe the Edge heuristic which we expect will be the basis of methods used in this stage in the future.

Since most problems are expected to be solved by stages 1 and 2, we shall describe the organization of these stages here. The control of the methods used in stage 3 is specific to these methods and will be described separately.

We note that the methods of stage 2 can call SIN to solve subproblems. When this occurs the flow of control and subproblems is given by Figure 2.

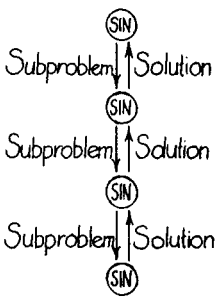


Figure 2 - Usual Flow of Control and Subproblems in SIN

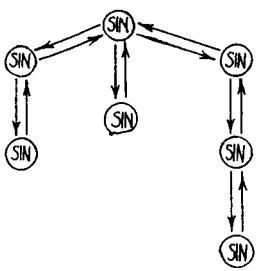


Figure 3 - Flow of Control and Subproblems in SIN When Problem is a Sum of Three Terms

If a subproblem is a sum, then each term in the sum will be integrated separately, and the flow is given by Figure 3.

It should be noted that if a method in stage 2 can transform a problem, the problem is not passed to another method in stage 2 or stage 3, even though the transformed problem cannot be integrated by SIN. For example,

$\int \sin(e^x)dx$ is transformed to $\int \frac{\sin y}{y} dy$ after substituting $y=e^x$ in stage 2. $\int \frac{\sin y}{y} dy$ cannot be integrated by SIN. Thus, SIN concludes that $\int \sin(e^x)dx$ is not integrable by it and will not pass it to stage 3.

In strictly enforcing such a decision we are depending upon the methods to employ tight progress requirements. If the progress requirements are made too tight, then few problems would be integrated by the methods of SIN's second stage. If, however, they are made too loose, then the methods of stage 2 would verify the hypothesis that they are applicable in problems in which they, in fact, are not appropriate, and thus SIN would fail to solve these problems. The experiments with SIN which are described in Appendices C, D, and E indicate the degree to which we succeeded in finding good progress requirements. We wish to point out that once such a discipline is successfully imposed on the methods, one is in a position to relax the requirement against backtracking, and thereby obtain somewhat greater power. We have not yet done so in SIN's second stage.

SAINT, in contrast to SIN's stages 1 and 2, will allow a problem to generate more than one subproblem. However, only one of the subproblems generated from any given problem must be solved in order to integrate the given problem. In general, the subproblems generated by SAINT during the

course of solution will form a tree structure. Figure 4 is a simplified description of the flow of control and subproblems in SAINT.

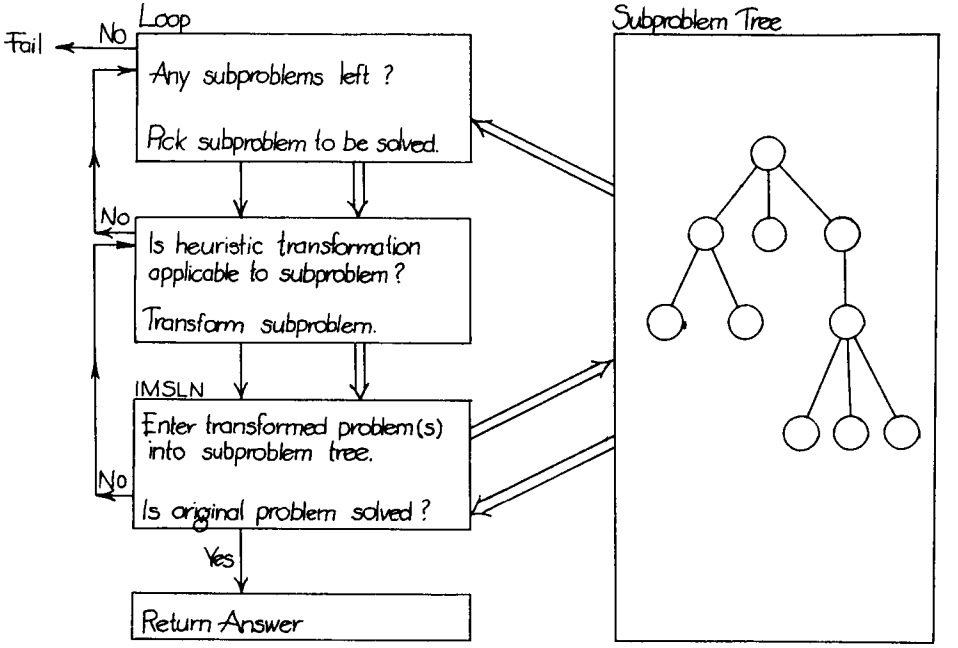


Figure 4 - Simplified flow of control (single arrow) and subproblems (double arrow) in SAINT

If a problem in SAINT generates more than one subproblem, the node in the tree corresponding to it is considered to be an OR node. Thus, only one of the subproblems must be solved. If the problem is a sum, a similar complication to the one in SIN is made. The node generated for such a problem is called an AND node. Each of the terms in the sum becomes a subproblem, and must be integrated. AND nodes are indi-

cated by an arc across the branches from that node. Thus, in general, a goal tree in SAINT has the form of Figure 5.

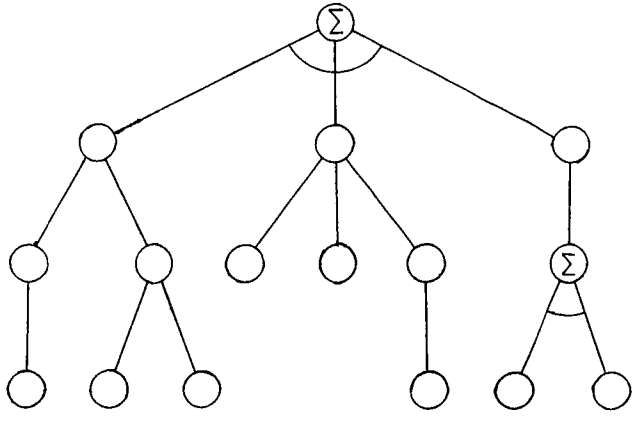


Figure 5 - A Subproblem Tree in SAINT when sums are present among the subproblems

All subproblems in SAINT are given to IMSLN. This includes the original problem and this fact is not shown in Figure 4. IMSLN thus acts like SIN's first stage. IMSLN has its own methods of solution. If it fails to solve the subproblem or some simple transformation of it, the subproblem will be put on the subproblem tree.

The routine LOOP (see Figure 4) has access to a list of subproblems to be tried called PLH. This list is ordered so that the first member of the list represents a subproblem which has the lowest depth of nested operators (e.g., PLUS, TIMES, COS) in the internal representation of the problem. LOOP will select the first subproblem on the list. It will

then ask each of the methods of SAINT called the heuristic transformations by Slagle to determine if they can transform the subproblem. These methods will be guided by information about the subproblem called the character of the subproblem. The character contains information such as whether the subproblem represents a rational function, an elementary function of exponentials or trigonemetric functions, etc. This information is used to limit the number of heuristic transformations applicable to a problem. Yet even with the use of the character mechanism as many as 11 out of the 17 heuristic transformations may be applied to a single subproblem.

The flow of control and information in SIN is called hierarchical. In a hierarchical organization, subproblems which are communicated between one routine and a second are private to these routines and are not known to the rest of the program. SAINT's organization can be called data base oriented. In such an organization the goal is to transform the data base (i.e., the goal tree in SAINT) to a desired state. In SAINT the desired state is a tree which has a path from the top node (the original problem) to a bottom node in which each node represents a solved problem. In a data base oriented organization control is relinquished to routines which manipulate the data base. In SAINT, all the heuristic transformations relinquish control to the IMSLN program.

SAINT's data base oriented approach allows and, in fact, may be said to encourage the program to backtrack, that is to leave one path of the tree and start on another. SIN's approach is to discourage backtracks at the first two stages. Backtracking is allowed in stage 3. However, in stage 3 backtracking is only of a limited nature.

Conventions

In describing SIN we shall use the usual convention that the variable of integration is x. SIN is actually a function of two arguments. The first is the expression to be integrated and the second is the variable of integration.

Below when we use the phrase "is a constant" we shall mean that the expression contains no occurrence of the variable of integration. Thus, $\sin^2 x + \cos^2 x$ is not a constant when x is the variable of integration.

We shall not concern ourselves here with difficulties which may arise due to the unsolvability of the constant or matching problem for the elementary functions. For a discussion of these difficulties see Appendix B.

By the elementary expressions of x we mean the set of expressions composed of

1) constants, 2) x, 3) trigonometric functions of x (e.g., sin(x), cos(x)), 4) logarithmic and arctrigonometric functions of x (e.g., $\log_e x$, arcsin x), and closed under the operations of addition, multiplication, exponentiation, and substitution.

By an elementary expression in f(x) (abbreviated elem(f(x)), we mean an expression obtained in the manner above, but where f(x) replaces x in the definition. Thus, for example, $(e^x + 1)e^{2e^x} + e^{2x}$ is an elementary expression of e^x. The expression xe^x, on the other hand, is an elementary expression of x, but not of e^x.

By a problem integrable in finite terms we mean a problem whose integral is representable by an elementary expression.

First Stage of SIN

The first stage of SIN uses the following three methods:

Method I If the integrand is a sum, each term is integrated separately by calling SIN iteratively and the results are added.

Method II If the integrand is of the form $[\Sigma u_i(x)]^n$, where n is a small positive integer, expand the expression and apply Method I.

Method III If the Derivative-divides routine is applicable, return its results.

The first two transformations are made so that the rest of the program can assume that the integrand is a product (though possibly a trivial product as in x or in e^x). The third method in this stage is the method which has led us to call this stage the stage that solves simple problems.

We shall now describe these methods in some detail.

I) Method I is an oft used method in practice. Using this method one avoids the difficulty of integrating dissimilar expressions such as $\sin x + e^x$. Integral tables, it will be noted, shun entries which are sums. However, this is not a safe rule to follow, in general. For example, let us consider $\int(e^{x^2} + 2x^2e^{x^2})dx$. Neither of the terms in this sum is completely integrable in terms of elementary functions. However, the sum is the derivative of xe^{x^2}. Hence, breaking up the terms in the sum and integrating them separately can disguise the integrability of the sum. This difficulty was known throughout the course of this research, and a heuristic for overcoming it in some cases was designed.*

* The heuristic that has been considered is of the following nature. Suppose we have a product of terms of the form f(x)g(x)h(x). The derivative is frequently of the form f'(x)g(x)h(x)+f(x)g'(x)h(x)+f(x)g(x)h'(x). Thus if one finds an integrand which is a sum such that two terms in the

However, no extension to this method has as yet been implemented. Slagle considered this method to be sufficiently safe so that he invariably followed it also.

Example

$$\int(\sin x + e^x)dx = \int \sin x\, dx + \int e^x\, dx$$

II) The reason for method II can be seen by considering the problem $\int(x + e^x)^2 dx$. SIN has no machinery which deals with this problem in its present form. However, if the problem is given as $\int(x^2 + 2xe^x + e^{2x})dx$, then the problem is easily integrated.

Example

$$\int(x + e^x)^2 dx = \int(x^2 + 2xe^x + e^{2x})dx$$

III) The Derivative-divides method is the heart of this stage in SIN. As we shall see many problems are integrated by it quite quickly. The inclusion of this method at this place in the program has an important methodological basis. It is presumed that in many computer problem solving systems there are methods of solution which solve most commonly occurring problems relatively quickly. If these methods are employed first by a problem solving system then many problems will be dispensed with in short order. Thus, the problem solving system will be able to afford to utilize expensive machinery in its later stages.

The Derivative-divides routine checks to see if the problem is of the form:

---

sum are related by having two factors in each of the forms f'g and fg', respectively, and with the rest of the factors identical, then one can guess the original product easily.

$$\int c\ op(u(x))u'(x)dx,$$

where c is a constant, u(x) is an elementary expression in x, u'(x) is its derivative, and op is an elementary operator. Op may be one of the following operators: a) identity b) sin c) cos d) tan e) cot f) sec g) csc h) arsin i) artan j) arsec k) log. Three more possibilities for op involve the exponentiation operation. These presume that the exponential function has only one nonconstant argument. Thus, we get the cases l) $u(x)^{-1}$ m) $u(x)^d$, $d \neq 1$, n) $d^{u(x)}$, where d is a constant. The final case is when the integrand is a constant and then u(x) is trivial. In that case the integral is simply cx.

The method of solution, once the problem has been determined to posses the form above, is to look up op in a table and substitute u(x) for each occurrence of x in the expression given in the table.* In other words, the method performs an implicit substitution y = u(x), and obtains the integral $\int c\ op(y)dy$ by a table look up.

Using this method the following examples can be integrated.

1) $\int \sin x \cos x\, dx = \frac{1}{2}\sin^2 x$, op = identity, u(x) = sin(x), u'(x) = cos(x), c = 1

2) $\int xe^{x^2} dx = \frac{1}{2}e^{x^2}$, op = $d^{u(x)}$, u(x) = x^2, u'(x) = 2x, c = $\frac{1}{2}$

3) $\int x\sqrt{1 + x^2}\, dx = \frac{1}{3}(1 + x^2)^{3/2}$, op = $u(x)^d$, u(x) = $1 + x^2$, u'(x) = 2x, c = $\frac{1}{2}$

4) $\int \frac{e^x}{1 + e^x} dx = \log(1 + e^x)$, op = $u(x)^{-1}$, u(x) = $1 + e^x$, u'(x) = e^x c = 1

---

* See Appendix A for a description of integral table look-up methods.

5) $\int x^{3/2}dx = \frac{2}{5}x^{5/2}$, op = $u(x)^d$, u(x) = x, u'(x) = 1, c = 1

A few more examples will indicate certain aspects of this method.

6) $\int \cos(2x + 3)dx = \frac{1}{2}\sin(2x + 3)$, op = cos, u(x) = 2x + 3, u'(x) = 2, $c = \frac{1}{2}$

The Derivative-divides method performs an implicit linear substitution in this case. SAINT would have performed an explicit linear substitution and would have required two calls to IMSLN to solve the problem.

7) $\int 2yze^{2x}dx = yze^{2x}$, op = $d^{u(x)}$, u(x) = 2x, u'(x) = 2, c = yz

This method handles constants easily. Constants can be generated or can be present in the integrand. SAINT would have removed the constants explicitly.

8) $\int \cos^2(e^x)\sin(e^x)e^x dx = -\frac{1}{3}\cos^3(e^x)$, op = $u(x)^d$, u(x) = $\cos(e^x)$, c = -1

This example demonstrates that the integral may be fairly complex and the method will still apply.

One of the experiments which was made with SIN was to attempt the 86 problems attempted by SAINT (see Appendix C). Interestingly enough, this method of Derivative-divides was able to solve fully 45 out of 86 problems. The average time on the 7094 was 0.6 seconds.

It is hoped that the above examples convincingly demonstrate the usefulness of this method at an early stage in an integration program. The method is to be recommended for those who desire an integration capability, but who are unable or unwilling to avail themselves of a more general program.

As was mentioned earlier, SAINT's IMSLN routine performs some

functions which are similar to SIN's first stage. IMSLN employs a table similar to that in the Derivative-divides routine but somewhat larger. It also performs eight transformations called algorithmic transformations by Slagle. These transformations are attempted one at a time. If one of them is successful the transformed problem is used and the original problem is not considered again. Two of these transformations are the same as method I and II in this stage of SIN. The others factor a constant or a negation operator from the integral; employ half angle identities; make a linear substitution; and perform certain simplifications on the integrand. As has been pointed out above, IMSLN also tends to the tree of subproblems and can determine if the original problem has been solved. IMSLN doesn't actually solve many problems so much as it is able to transform a great number of problems into a form which is more easily solved by the rest of SAINT. It would appear that SIN's Derivative-divides method solves more problems immediately than does IMSLN. SAINT's Derivative-divides heuristic transformation, which is quite powerful, is not applied to a problem until much later in the course of the solution.

The Second Stage of SIN

If a problem fails to be solved by SIN's first stage, then it is determined whether one of eleven additional methods is applicable to it. In order to determine which method is to be applied clues are obtained from the expression. We have called the technique by which these clues are used hypothesis formation (see Chapter 2). The routine that obtains these clues and conducts the formation of an hypothesis is called FORM. Associated with most of the methods are patterns in SCHATCHEN

which serve to differentiate the problems which are solvable by each method from those solvable by other methods. It turns out that few problems have more than one method applicable to them. In the cases where a conflict does exist (e.g., in solving problems with algebraic integrands) the actual method chosen appears to have little effect on the cost of obtaining a solution.

In this stage of SIN, a single method (Method 6) handles problems which involve trigonometric expressions. When FORM sees a subexpression of an integrand which is a trigonometric function of a linear argument in the variable of integration, this subexpression will act as a clue, and FORM will call Method 6 to validate the hypothesis that a substitution can be made for the trigonometric functions. If Method 6 decides that such a substitution is not applicable (e.g., $\int \sin x \, e^x dx$), then it will return the value NIL (FALSE). In such a case, FORM might entertain another hypothesis but since there are none for trigonometric functions, FORM will also return the value NIL. If Method 6 finds that a transformation is applicable, it will hand SIN the transformed problem. The value of SIN, with a proper substitution to account for the transformation that was made will be returned as the value of Method 6 and of FORM.

Examples of problems integrated by this stage of SIN:
(It is probable that none of these could be integrated by SAINT.)

1) $\displaystyle\int \frac{\sqrt{A^2 + B^2 \sin^2 x}}{\sin x} \, dx$

2) $\displaystyle\int (1 + 2x^2) e^{x^2} dx$

3) $\int \frac{e^{2x}}{A + Be^{4x}}\, dx$

4) $\int x\sqrt{x + 1}\, dx$

5) $\int x^{1/2}(x + 1)^{5/2} dx$

6) $\int \frac{1}{x^4 - 1}\, dx$

Below we describe each of the methods used in this stage. Each description contains the clue which FORM uses to determine whether the method might be applicable. A more extended description of the manner in which FORM operates will then follow.

Method 1) Elementary function of exponentials.

This method is applicable whenever the integrand has the form of an elementary function of $a_i^{b_i x + c_i}$, where the a_i, b_i, and c_i are constants.

Clue - a subexpression of form a^{bx+c}; a, b, c are constants.

Examples -

$\int \frac{e^x}{2 + 3e^{2x}}\, dx$ becomes $\int \frac{1}{2 + 3y^2}\, dx, \quad y = e^x$

$\int \frac{e^{2x}}{A + Be^{4x}}\, dx$ becomes $\int \frac{y}{A + By^4}\, dy, \quad y = e^x$

$\int \frac{e^{x+1}}{1 + e^x}\, dx$ becomes $\int \frac{e}{1 + y}\, dy, \quad y = e^x$ and $e^{x+1} = ee^x$

$\int 10^x e^x dx$ becomes $\int y^{\log e^{10}}\, dy, \quad y = e^x$

Method - $a_i^{b_i x+c_i}$ is transformed into $a_1^{(b_i x+c_i)\log_{a1} a_i}$ in order to convert all bases to a common base a_1. Here a_1 is the first base encountered in the integrand.

a_1^{bx+c} where $c \neq 0$ is converted to $a_1^c a_1^{bx}$. This facilitates the transformation to be made.

The substitution $y = a_1^x$ is made. Thus, each a_1^{bx} is replaced by y^b and the resulting expression is divided by $y \log_e a1$.

Notes - What is controversial about this method is that in converts all bases to a single base which in not necessarily e. This may lead to the introduction of unnecessarily clumsy constants (e.g., $\log_5 3$).

SAINT's method in this case was somewhat different. SAINT did not handle different bases, nor all cases where constants (i.e., c_i) were present in the exponent. It did, though, find the greatest common divisor of the b_i, k, say, and made the substitution $y = a_1^{kx}$. In SIN this will be handled by algorithm 2 which will make the substitution $z = y^k$ after $y = a_1^x$ is made by the current method. The method that performs the substitution $z = y^k$ was not present in SAINT although it was suggested as an extension

Method 2) Substitution for an integral power.

This method is applicable whenever the integrand is of the form

$x^c \text{ Elem}(x^{k_i})$, where c, k_i are integers and where

$k = \gcd(\{c + 1, k_1, k_2, \ldots\}), k \neq 1$

Clue - Instead of obtaining a clue which determines whether this transformation is applicable, FORM obtains a clue which determines whether this transformation is not possible. FORM will note that this transformation is not applicable when it sees a subexpression of the

form e^{a+bx} or sin(x). If none of the other methods is applicable, and no such clue has been found, this transformation will be called.

Examples -

$\int x^3 \sin(x^2)dx$ becomes $\int \frac{1}{2} y \sin y \, dy, \; y = x^2$

$\int \frac{x^7}{x^{12} + 1} dx$ becomes $\int \frac{1}{4} \frac{y}{y^3 + 1} dy, \; y = x^4$

Method - Substitute $y = x^k$

Notes - This method was suggested but not implemented by Slagle who embedded it in a larger method which was implemented in SIN in two separate methods (2 and 3).

This method is currently restricted to integer exponents. It should be extended to handle exponents such as 3a, 2a in

$$\int x^{3a} \sin(x^{2a})dx$$

Method 3) Substitution for a rational root of a linear fraction of x.

This method is applicable when the integrand is of the form

$$\text{Elem}(x, \left(\frac{ax + b}{cx + d}\right)^{\frac{n_1}{m_1}}, \left(\frac{ax + b}{cx + d}\right)^{\frac{n_2}{m_2}}, \ldots)$$

where the n_i and m_i are relatively prime integers with some $|m_i| \neq 1$, and with a, b, c, d constants and $ad - bc \neq 0$.

Clue - A subexpression of the form

$\left(\frac{ax + b}{cx + d}\right)^{\frac{n}{m}}$ a, b, c, d constants; n, m, relatively prime integers, $|m| \neq 1$

Examples -

$\int \cos \sqrt{x} \, dx$ becomes $\int 2y \cos y \, dy, \; y = \sqrt{x}$

$\int x\sqrt{x+1}\,dx$ becomes $\int 2(y^2 - 1)y^2 dy,\ y = \sqrt{x+1}$

The above two problems were attempted and not solved by SAINT.

$\int \frac{1}{x^{1/2} - x^{1/3}}\,dx$ becomes $\int 6y^5 \frac{1}{y^3 - y^2}\,dy,\ y = x^{1/6}$

$\int \sqrt{\frac{x+1}{2x+3}}\,dx$ becomes $\int \frac{2y^2}{(2y^2 - 1)^2}\,dy,\ y = \sqrt{\frac{x+1}{2x+3}}$

Method - Let k = least common multiple of the m_i.

Substitute $y = \left(\frac{ax + b}{cx + d}\right)^{1/k}$

Notes - The restriction ad - bc $\neq$ 0 assures that the substitution is non-trivial. If ad - bc = 0, then $\frac{dy}{dx} = 0$.

Slagle suggested methods 2 and 3 as a single method. Considering them as two separate methods facilitated the coding. This method is an extension of Slagle's suggestion since it covers linear functions.

Even this algorithm should be split into two parts. One would handle the case restricted to $(ax + b)^{n/m}$, the other the more general case $\left(\frac{ax + b}{cx + d}\right)^{n/m}$.

Much of the time only the former is needed, but the machinery for handling the latter, which is more expensive, is employed.

A weakness of this routine is its inability to deal with variable exponents. These would usually result in the generation of a reduction formula as opposed to an integral. The great advantage of an integral table over SIN currently is the presence of the reduction formulas. The Edge heuristic (See Chapter 5) can generate some reduction formulas,

but not many at present. (Or course, an instance of a variable exponent should result in a solution in SIN!)

Method 4) Binomial - Chebyschev

This method is applicable whenever the integrand is not a rational function and possesses the form

$Ax^r(c_1 + c_2x^q)^p$, where A, c_1, c_2 are constants, p, q, r are rational numbers and $c_1c_2qp \neq 0$.

Clue - A subexpression which is a nonintegral power of a rational function. This is followed in FORM by a match of the integrand and the form above.

Examples

$\int x^4(1 - x^2)^{-5/2}dx$ becomes $\int \frac{-1}{y^4(1 + y^2)}\,dy, \quad y = \frac{\sqrt{1 - x^2}}{x}$

$\int x^{1/2}(1 + x)^{5/2}dx$ becomes $\int \frac{-2y^6}{(y^2 - 1)^5}\,dy, \quad y = \sqrt{\frac{x + 1}{x}}$

Method - Binomial conversion to Chebyschev form (substitute $y = x^q$). Thus $A \leftarrow A/q$, and $r_2 \leftarrow p$, $r_1 \leftarrow \frac{r + 1}{q} - 1$

Make the first applicable transformation

a) r_1 integer, $r_2 > 0$

Substitute $z = c_1 + c_2 y$

b) r_2 integer, r_1 a rational number with denominator d_1

Substitute $z = y^{1/d1}$

c) r_1 integer, $r_1 < 0$, r_2 rational number with denominator d_2

Substitute $z = (c_1 + c_2y)^{1/d_2}$

d) $r_1 + r_2$ is an integer

Substitute $z = \left(\frac{c_1 + c_2 y}{y}\right)^{1/d_1}$

Otherwise, return notice of failure to integrate problem.

Notes - This method was also suggested but not implemented by Slagle. It has the advantage of being a decision procedure. That is, if an integrand has the form given above, then either the method yields the integral or the problem cannot be integrated in finite terms. This was proved by Chebyschev (see Ritt [54], p. 27).

The argument used is roughly as follows: If r_1, r_2, or $r_1 + r_2$ is an integer, then the substitutions above result in rational functions and thus can be integrated. Otherwise we know from Abel's Theorem (see Chapter 5) that the integral, if it is expressible in finite terms, is a sum of an algebraic function and logarithmic terms. The residue of a Chebyschev function is everywhere 0. Hence the integral cannot contain logarithmic terms. Further analysis shows that the assumption that the integral is algebraic leads to a contradiction.

In this case also the integral tables contain many entries which are reduction formulas for the cases when p, q, r are parameters. Some such capability should be present in SIN also.

Method 5) Arctrigonometric substitutions

This method is applicable whenever the integral is of the form $R(x, \sqrt{cx^2 + bx + a})$ where a, b, c are constants and R is a rational function of its arguments.

Clue - A subexpression of the form $(cx^2 + bx + a)^{n/2}$, where n is an odd integer.

Examples

$$\int \frac{x^4}{(1 - x^2)^{5/2}}\,dx \quad \text{becomes} \quad \int \frac{\sin^4 z}{\cos^4 z}\,dz, \quad y = \arcsin x$$

$$\int \frac{\sqrt{A^2 + B^2 - B^2y^2}}{1 - y^2}\,dy \quad \text{becomes} \quad \int \frac{1}{B} \frac{(A^2 + B^2)\cos^2 z}{(1 - \frac{A^2 + B^2}{B^2}\sin^2 z)}\,dz, \quad z = \arcsin \frac{By}{\sqrt{A^2 + B^2}}$$

Method

First eliminate the middle term of the quadratic by completing the square

$$y = x + \frac{b}{2c},$$

yielding the integrand in the form

$$R(y - \frac{b}{2c}, \sqrt{cy^2 + a - \frac{b^2}{4c}})$$

Let $A = a - \frac{b^2}{4c}$

$C = c$

If $C > 0$, $A > 0$, substitute $z = \arctan \sqrt{\frac{C}{A}}\, y$

If $C > 0$, $A < 0$, substitute $z = \arcsin \sqrt{\frac{C}{-A}}\, y$

If $C > 0$, $A = 0$, replace the quadratic by $\sqrt{C}\, y$

If $C < 0$, $A > 0$, substitute $z = \text{arcsec} \sqrt{\frac{-C}{A}}\, y$

If A and C are both numbers, then the signs are determined trivially. If A or C are parameters, then the user will be asked whether they are positive, negative, or zero through an appropriate message at the console. For example if the value of A is e, a message would read

IS e POSITIVE

An answer of "yes" is expected if e is in fact positive. However, the program can frequently determine whether A or C are positive. This is

done by assuming that all parameters are real valued and by using the fact that sums of squares of real numbers are positive. Thus

$$2d^2 + 3e^4 + 5$$

is determined to be positive, whereas

$$-d^2 - 2(e + f)^2$$

is determined to be negative. A single SCHATCHEN rule is used in making this determination.

Notes

In cases where the coefficients are parameters, it is possible to run the program several times and answer questions differently each time.

SAINT had two transformations which performed the function of this method. One method eliminated the middle term from all quadratics, another made the arctrigonometric substitutions in all quadratics with missing middle terms. The arctrigonometric substitutions are normally made for roots of quadratics as we have done and not in all quadratics as SAINT attempted to do. SAINT also implicitly required that the coefficients in the quadratic be numbers. The kind of interaction between the user and the program which is required when one allows nonnumerical coefficients became practical when time-sharing systems were introduced.

Method 6) Elementary function of trigonometric functions.

This method is applicable when the integrand is an elementary function of the trigonometric functions applied to linear argument in the variable of integration.

Clue - TRIG(a + bx) where TRIG $\in$ {sin, cos, sec, tan, cot, csc}

Examples

1) $\int \sin^2 x\, dx$ becomes $\int (\frac{1}{2} - \frac{1}{2}\cos 2x)dx$

2) $\int \frac{\sqrt{A^2 + B^2\sin^2 x}}{\sin x}\, dx$ becomes $\int \frac{\sqrt{A^2 + B^2(1 - y^2)}}{1 - y^2}\, dy$, $y = \cos x$

3) $\int \frac{1}{1 + \cos x}\, dx$ becomes $\int dy$, $y = \tan \frac{1}{2}x$

Method

I) In problems where the arguments of the trigonometric functions are not the same throughout the integrand, the following cases are examined.

a) $\int \sin m x \cos n x\, dx = \frac{-\cos(m - n)}{2(m - n)} x - \frac{\cos(m + n)x}{2(m + n)}$

b) $\int \sin m x \sin n x\, dx = \frac{\sin(m - n)x}{2(m - n)} - \frac{\sin(m + n)x}{2(m + n)}$

c) $\int \cos m x \cos n x\, dx = \frac{\sin(m - n)x}{2(m - n)} + \frac{\sin(m + n)x}{2(m + n)}$ m, n, constants $m \neq -n$

Otherwise, the method returns notice of failure to integrate the problem.

II) If the arguments are the same but are not identically x, a linear substitution $y = a + bx$ is performed and the procedure continues with the revised problem.

III) If the problem is of the form

$\int \sin^m(y)\cos^n(y)dy$; m, n integers

a) $m < n$, transform to $\int (\frac{1}{2}\sin 2y)^n (\frac{1}{2} + \frac{1}{2}\cos 2y)^{\frac{n - m}{2}}\, dy$

b) $m \geq n$, transform to $\int (\frac{1}{2}\sin 2y)^n (\frac{1}{2} - \frac{1}{2}\cos 2y)^{\frac{m - n}{2}}\, dy$

IV) All trigonometric functions are transformed into sines and cosines (e.g., $\tan y \rightarrow \frac{\sin y}{\cos y}$) in order to test if the resulting expression is of the form a or b.

a) $\int \sin^{2n+1}(y)\,\mathrm{Elem}(\sin^2(y),\cos(y))\,dy.$

In this case substitute $z = \cos(y)$

b) $\int \cos^{2n+1}(y)\,\mathrm{Elem}(\cos^2(y),\sin(y))\,dy$

In this case substitute $z = \sin(y)$.

V) All trigonometric functions are transformed into secants and tangents in order to test whether the resulting expression is of the form:

$$\int \mathrm{Elem}(\tan(y),\sec^2(y))\,dy$$

In this case substitute $z = \tan(y)$.

VI) Finally, the substitution $z = \tan\frac{1}{2}y = \frac{\sin y}{1 + \cos y}$ is made.

Notes - In the case where the integrand is a rational function of trigonometric functions of x all the problems can be reduced to rational functions. The choice of the transformation governs the simplicity of the resulting rational function and the final answer. The transformation in step VI above which is always applicable in these situations frequently leads to a great deal of work and to complex results. Fortunately, a number of simpler transformations such as those of steps III, IV, and V are easily recognized and are usually applicable.

SAINT included all of the transformations given above, but they were embedded in different places in the program. I is included in the integral table. II is an algorithmic transformation, as is step III.

IV and V are three separate heuristic transformations. V is yet another heuristic transformation. The initial stage in steps IV and V is performed by still another method. This organization of the methods applicable to trigonometric functions led to the generation of extraneous subproblems since the heuristic transformations were disjoint and were not aware of each others actions, nor, in fact, of their own actions. For example, the method which performs the preliminary transformation in steps IV and V (e.g., tan x $\rightarrow \frac{\sin x}{\cos x}$) must be inhibited from performing the opposite transformation later (e.g., sin $\rightarrow \frac{\tan}{\sec}$).

More work is necessary in this area in handling arguments to trigonometric functions which are linear, but different (e.g., $\int \frac{\sin(7x)}{\cos(6x)} dx$). Some programs along this line have been designed by Edmund Berkeley, but they have not been fully implemented.

Method 7) Rational function times an exponential

This method is applicable whenever the integrand is of the form $R(x)e^{P(x)}$, where R(x) is a rational function in x and P(x) is a polynomial in x.

Clue - A subexpression of the form $e^{P(x)}$, where P is a polynomial in x. If P(x) is linear in x, this method will be attempted if method 1 is not applicable.

Examples

1. $\int xe^x dx = xe^x - e^x$
2. $\int \frac{x}{(x+1)^2} e^x dx = \frac{e^x}{x+1}$
3. $\int (1 + 2x^2)e^{x^2} dx = xe^{x^2}$
4. $\int e^{x^2} dx$: not integrable

5. $\int \frac{e^x}{x}\,dx$: not integrable

Method - This method once again is a decision procedure. That is, the method can tell whether a problem can be integrated in finite terms or not. The method is an improvement of the decision procedure in Ritt [54](p. 48) which handled the case by solving a system of linear equations. The procedure is an application of the Liouville theory for integration about which more will be found in Chapter 5. This procedure is similar in flavor to Risch's [53] recent theoretical treatment of results in the Liouville theory.

Let $R(x) = \frac{C_1x^{m_1} + S_1(x)}{Q(x)}$ where S_1, Q are polynomials

S_1 is a polynomial of degree $< m_1$,

C_1 is a constant, $C_1 \neq 0$.

We know from the Liouville theory that the integral (if any) will be a multiple of $e^{P(x)}$. (See Ritt [54], page 47.)

Suppose the integral is represented by

$$(a_1(x) + b_1(x))e^{P(x)}, \quad \text{then}$$

$$P'(x)a_1 + a_1' + P'(x)b_1 + b_1' = R(x) = \frac{C_1x^{m_1} + S_1(x)}{Q(x)}$$

Let $a_1(x) = \frac{C_1x^{m_1}}{P'Q}$, then

$$a_1' = \frac{m_1C_1x^{m_1 - 1} - \frac{C_1x^{m_1}Q'}{P'Q} - \frac{C_1x^{m_1}P''}{(P')^2}}{Q}$$

and

$$P'b_1 + b_1' = R(x) - P'a_1 - a_1' = \frac{S_1 - \frac{m_1C_1x^{m_1 - 1}}{P'} + \frac{C_1x^{m_1}P''}{(P')^2} + \frac{C_1x^{m_1}Q}{P'Q}}{Q}$$

The numerator of $P'b_1 + b_1'$ is a polynomial $T_1(x)$, say, and a rational function remainder, $U_1(x)$, say. Let the leading term of $T_1(x)$ be $C_2x^{m_2}$ and

the rest of $T_1(x)$ be $S_2(x)$. Now continue the process indicated above until some T_i (T_n, say) is 0. This is guaranteed to occur since the degree of the T_i is decreasing. If at that time the corresponding U_i (i.e., U_n) is also 0, then the expression $R(x)e^{P(x)}$ is integrable and the integral is $\sum_{i=1}^{n} a_i(x)e^{P(x)}$. If U_n is not 0, then the problem is not integrable in finite terms.

Note that if $U_n = 0$, then $R(x) - P' \sum_{i=1}^{n} a_i - \sum_{i=1}^{n} a_i' = 0$.

Let $a = \sum_{i=1}^{n} a_i(x)$; then we obtain the relation

$$P'a + a' = R$$
$$P'ae^P + a'e^P = Re^P$$
$$(ae^P)' = Re^P$$
$$ae^P = \int Re^P dx$$

For the converse, we refer to Ritt. Also, note the discussion in Chapter 5.

Notes - SAINT was able to solve the first two of the examples above. Both were solved using the Integration-by-parts method of SAINT.

SAINT was unable to integrate $\int e^{x^2} dx$ because it found that no transformations were applicable to the problem after approximately one minute of computation.

The fact that SAINT was unable to integrate this problem does not necessarily mean that the problem is not integrable in finite terms. This statement is also true of SIN, in general. This is due to the fail-safe nature of the programs. When a fail-safe integration program results in

an integral then we know that the problem is integrable. When such a program does not yield an integral then one still does not know whether the problem can be integrated or not. A semi-decision procedure for integration would, in finite time, result in an integral or in the statement that the problem cannot be integrated in finite terms. Richardson's result (see Appendix B) shows that for the integration problem as he defines it, no decision procedure exists. Yet decision procedures exist for many interesting subcases and especially when one avoids considering the matching problems that Richardson shows are inherent in his characterization of the elementary functions. SIN embodies some decision procedures. Future programs are likely to contain more (see Chapter 5). One must be quite careful about the computational methods involved in order to avoid the explosion which is apparently inherent in many decision procedures in algebraic manipulation (see Moses[42]). We would prefer to see expensive decision methods to be attempted as a last resort, such as stage 3 in SIN. A final consideration regarding methods for integration is that they should not be too radical or else the result will become less meaningful to the human user.

This method was implemented using the rational function package of MATHLAB [36]. SIN communicates with the rational function package by a process called chaining. More will be said about chaining when we discuss the integration of rational functions.

Method 8) Rational functions

This method is applicable whenever the integrand is a rational function.

Clue - FORM generates no local clue for rational functions. The applicability of this method is determined separately. Sometimes this

method is called directly by other methods (e.g., methods 2 and 4).

Examples

1. $\int \frac{x}{x^3 + 1}\, dx = -\frac{1}{3}\log_e (x + 1) + \frac{1}{6}\log_e (x^2 - x + 1) + \frac{1}{\sqrt{3}} \arctan \left(\frac{2x - 1}{\sqrt{3}}\right)$

2. $\int \frac{1}{x^6 - 1}\, dx = -\frac{1}{6}\log_e (x + 1) + \frac{1}{6}\log_e (x + 1) + \frac{1}{12}\log_e (x^2 - x + 1) -$
$- \frac{1}{2\sqrt{3}} \arctan\left(\frac{2x - 1}{\sqrt{3}}\right) - \frac{1}{12}\log_e (x^2 + x + 1) - \frac{1}{2\sqrt{3}} \arctan\left(\frac{2x + 1}{\sqrt{3}}\right)$

3. $\int \frac{1}{(B^2 - A^2)x^2 - A^2B^2 + A^4}\, dx = \frac{-1}{2AB^2 - A^3} \log_e (x + A) + \frac{1}{2AB^2 - A^3} \log_e (x - A)$

Method - This method was programmed for the MATHLAB system by Manove and Bloom under the direction of Engelman of the MITRE Corporation. The integration procedure which is used is described in Hardy [25]. The polynomial factorization routine used in this program essentially follows Kronecker's method as described in Van der Waerden [65], p. 77-78. This program is also written in LISP and is itself described in "Rational Functions in MATHLAB," by Manove, Bloom and Engelman [36].

Notes - The power of this method makes the coding of the rest of SIN a great deal simpler. SAINT did not have a powerful rational function integration program (it could integrate polynomials and ratios of polynomials with linear and quadratic factors) and it suffered thereby; much of the trial and error in some problems for SAINT can be attributed to its inability to integrate certain rational functions which arose as subproblems. Some of the extensions which were made to SAINT (e.g., methods 2 and 4) could not have been made unless a rational function program was present. Thus, the second stage of SIN lets this routine clean up the details such as rationalization of denominators which could be ignored in making the transformations.

Slagle realized that the unavailability of a rational function integration program was a basic defect in SAINT. However his proposal for the manner in which such a routine should be written was not the best. He proposed solving linear equations to obtain a partial faction expansion of the rational function. The method in MATHLAB is superior computationally.

As was mentioned earlier the experimental work (e.g., debugging and testing) was done using Project MAC's time sharing system CTSS. One valuable feature of this system is the power to use programs written by others. In our case it was valuable to have access to the rational function package of the MATHLAB system. To be sure, in conventional "batch" processing one can employ large packages designed by others by using intermediate tapes. In CTSS one can conveniently make use of a program concurrently under development by another group, providing one is prepared to spend some time for the process involved.*

The rational function program which SIN uses is available in CTSS as FULMAN SAVED. It is a separate core image from SIN and is called using the chaining process given below.

a) SIN writes the problem to be integrated on file MANOVE LISP.

b) SIN saves itself as MOSES SAVED.

---

* The widespread availability of time sharing consoles has allowed SIN to be used by several people other than the author. "Bugs" in the program have been pointed out by Michael Levin of Information International, Inc., Carl Hewitt and Peter Samson of Project MAC, and Russel Kirsch of the National Bureau of Standards. We would hereby like to express our appreciation of their efforts.

c) SIN directs CTSS to execute FULMAN SAVED.

d) FULMAN reads MANOVE LISP.

e) FULMAN generates a solution to the problem.

f) FULMAN writes the solution on file MANOVE ANS.

g) FULMAN directs CTSS to resume MOSES SAVED.

h) MOSES (i.e., SIN) reads MANOVE ANS.

Experimentally the minimum time for this process has been determined to be about 4.5 seconds of machine time. Most of the time is spent in steps c and g in which 32k programs are loaded into core.

There are, at present, certain differences in the internal representation used in SIN and FULMAN. These differences are eliminated, whenever possible, by two interface routines present in SIN. The differences consist of the following:

a) log has two arguments in SIN, one in FULMAN.

b) PLUS, TIMES have variable number or arguments in SIN and only two in FULMAN.

c) No floating point numbers are allowed in FULMAN. Whenever possible these are converted to rational numbers (i.e., (a·b) where a,b are integers). Otherwise an error indication is given in SIN.

Method 9) Rational function times a log or arctrigonometric function with a rational argument.

This method is applicable whenever the integrand is of the form

$R(x)F(S(x))$ where F is log, arcsin, or arctan

$R(x)$ and $S(x)$ are rational functions

and where $\int R(x)dx$ is also rational.

Clue - F(S(x)) where F is log, arcsin or arctan and S(x) is a rational function.

Examples

1) $\int x \log_e x \, dx$ becomes $\frac{x^2}{2}\log_e x - \int \frac{x}{2} \, dx$

2) $\int x^2 \arcsin x \, dx$ becomes $\frac{x^3}{3}\arcsin x - \int \frac{x^3}{3\sqrt{1 - x^2}} \, dx$

3) $\frac{1}{x^2 + 2x + 1}\log(x^2 + 2x)$ becomes $\frac{-1}{x + 1}\log(x^2 + 2x) - \int \frac{-1}{x + 1} \frac{(2x + 2)}{x^2 + 2x} \, dx$

Method - Let $T(x) = \int R(x)dx$

a) F = log

Solution is $T(x)\log(S(x) - \int T(x) \frac{S'(x)}{S(x)} \, dx$

b) F = arctan

Solution is $T(x)\arctan S(x) - \int T(x) \frac{S(x)}{1 + S^2(x)} \, dx$

c) F = arcsin

Solution is $T(x)\arcsin S(x) - \int T(x) \frac{S'(x)}{\sqrt{1 - S^2(x)}} \, dx$

Notes - This routine handles three special cases of the method of Integration-by-parts. The utility of these special cases is that they direct the solution process quite clearly, whereas the more general solution methods may make false starts or require more extended analysis.

SAINT would have attempted to solve most of the problems that fall under this category with its Integration-by-parts method. If we presume that SIN had only the rational function capability of SAINT, then this method would allow SIN to be more powerful on these problems to which this method applies. This is due to the fact that SAINT could not tell how much effort it could reasonably expend on its Integration-by-parts method and it chose to spend less effort of it than would be required to integrate the third problem avove, for example.

Method 10) Rational function times an elementary function of $\log_c(a + bx)$.

This method is applicable whenever the integrand is of the form $R(x)\text{Elem}(\log_c(a + bx))$ where $R(x)$ is a rational function and a, b, c, are constants.

Clue - A subexpression of the form $\log_c(a + bx)$. This method is attempted if method 9 fails to be applicable.

Examples

1) $\int \frac{\log_e x}{(\log_e x + 1)^2}\,dx$ becomes $\int \frac{y}{(y + 1)^2}\,e^y dy,\ y = \log_e x$

2) $\int \frac{1}{x}\,\frac{1}{1 + \log_e^2 x}\,dx$ becomes $\int \frac{1}{1 + y^2}\,dy,\ y = \log_e x$

3) $\int \frac{1}{\log_e x}\,dx$ becomes $\int \frac{1}{y} e^y dy,\ y = e^x$

Method - Substitute $y = \log_c(a + bx)$

results in

$$\int R\left(\frac{c^y - a}{b}\right)\text{Elem}(y)\frac{c^y}{b}\log_e c\,dy$$

Notes - This method is used to reduce the problem to the exponential case where the powerful method 7 might be applicable. If method 7 is not applicable, the transformed problem stands as much a chance of being integrated by SIN's current methods as did the original problem in the logarithmic form.

Method 11) Expansion of the integrand.

This method is applicable whenever the integrand can be expanded by distributing sums over products.

Clue - This method is used whenever all of the previous methods have failed to be applicable. No clue for the applicability of this method is found by FORM.

Examples

$\int x(\cos x + \sin x)dx$ becomes $\int (x \sin x + x \cos x)dx$

$\int \frac{x + e^x}{e^x} dx$ becomes $\int (xe^{-x} + 1)dx$

$\int x(1 + e^x)^2 dx$ becomes $\int (x + 2xe^x + xe^{2x})dx$

Notes - SAINT had two heuristic transformations which together performed the job of this method. The first distributed nonconstant sums in products, the second expanded positive integer powers of nonconstant sums. In both cases, where Slagle considered the methods inappropriate, SIN would have already applied one of the previous methods and solved the problem. As a matter of fact, that is also true of the two problems for which he considered the methods to be appropriate.

The Third Stage of SIN

This stage, the last stage of SIN, is the appropriate place for methods of a rather general nature.

Two methods which properly belong in this stage have been programmed. The first is the Integration-by-parts method. This method is used in the experiment in Appendix C in which SIN was asked to solve the 86 problems attempted by SAINT. Only two of those problems (i.e., $\int x \cos x\, dx$ and $\int \cos \sqrt{x}\, dx$) required this method. The second method is based on the Edge heuristic described in Chapter 5. A third method, a powerful Derivative-divides method, has not been implemented, but will be discussed here.

In the long run it is expected that only one of these methods will be used here--that is the method based on the Edge heuristic or some vari-

ant of it.

The Integration-by-Parts Method

Examples -

1) $\int x \cos x \, dx$ becomes $x \sin x - \int \sin x \, dx$

2) $\int x \log_e^2 x \, dx$ becomes $\frac{x^2}{2}\log_2^2 x - \int x \log_e x \, dx$

Method - Let Maxparts be twice the maximum of the value of a constant exponent of any nonconstant factor in the integrand. Thus Maxparts is 2 for x cos(x) and 4 for $x^2 \cos x$.

Consider any partition of the integrand into a product of nonconstant factors g and h, where $H(x) = \int h \, dx$ can be obtained by SIN without calling the Integration-by-parts method.

Now consider $\int g'H dx$. We require that this integral be found by SIN by calling the Integration-by-parts method fewer than Maxparts times.

If both integrals are obtained, the solution is

$$\int gh \, dx = gH - \int g'H \, dx.$$

Notes - The crucial aspect of this method is embodied in the phrase "consider any partition." This method is thus willing to attempt several partitions of the integrand. It is thus searching for a solution, and searching very blindly indeed.

In order to avoid searching too large a space, we require that H(x) must be found without using this method. SAINT, which also had an Integration-by-parts method required that H(x) be found by IMSLN, which is a stronger restriction. Likewise the Maxparts device avoids an infinite search for the second integral. SAINT, which did not use such a device

appears vulnerable to difficulties such as in the problem $\int \frac{\sin x}{x}\,dx$.

Consider $h = \sin x$, $g = \frac{1}{x}$. Thus $\int h\,dx = -\cos x$ and $\int g'H\,dx = \int \frac{\cos x}{x^2}\,dx$.

One subproblem generated by $\int \frac{\cos x}{x^2}\,dx$ is $\int \frac{\sin x}{x^3}\,dx$. This process can continue indefinitely unless measures are taken to curtail it. (Actually $\int \frac{\sin x}{x}\,dx$ is not integrable in finite terms.)

The Derivative-Divides Method

The method of Integration-by-parts and the Derivative-divides method are the two general methods that a freshman calculus student is likely to learn. Let us recall that SIN's first stage employed a Derivative-divides method. However, that method is not as general as it might be. The Derivative-divides method attempts to determine whether the integrand can be put into the form $g(u(x))u'(x)$. If this is the case then the substitution $y = u(x)$ transforms the problem into $\int g(y)dy$. In stage 1, g was required to be a single operator. However, in a more general method g would not be so limited and the following problems would be transformed by this method. (Let us note again that this method is not available in SIN at present.)

1) $\int \cos x(1 + \sin^3 x)dx$ becomes $\int (1 + y^3)dy$, $y = \sin x$

2) $\int \frac{1}{x} \frac{1}{1 + \log_e^2 x}dx$ becomes $\int \frac{1}{1 + y^2}\,dy$, $y = \log_e y$

3) $\int \frac{1}{\sqrt{1 - x^2}} \frac{1}{1 + \arcsin^2 x}\,dx$ becomes $\int \frac{1}{1 + y^2}\,dy$, $y = \arcsin x$

The first two of these problems can be solved by SIN's second stage (in particular by methods 6 and 10). The third problem is one of the simplest examples of a problem which cannot be solved by SIN's first two

stages along with the Integration-by-parts method. However, the Edge heuristic will correctly guess the integral arctan(arcsin x).

SAINT had a Derivative-divides method which was more powerful than SIN's. However, it suggested many bad transformations in some cases. The method essentially performed a search for a subexpression such that the number of factors in the quotient of the expression and the derivative of the subexpression was smaller than the number of factors in the original integrand. This is too strong a restriction sometimes.

A Derivative-divides method was designed but was never implemented in SIN.

The kind of analysis we considered was as follows: Suppose the integrand is f(x) and a nonlinear subexpression of it is u(x), then if $\frac{f(x)}{u'(x)}$ can be represented as g(u(x)), the method would propose substituting y = u(x) and attempting $\int g(y)dy$. We should, though, restrict the kind of functions g that we would allow. For example, in $\frac{\sin x}{\sin x + \cos x}$ we might wish to disallow the substitution y = cos x since it introduces the algebraic term $\sqrt{1 - y^2}$ into the denominator. If we make the conditions on the g's sufficiently restrictive (e.g., rational, algebraic) then the number of substitutions per problem that this method would propose would be small, and more significantly, each of the substitutions would be quite reasonable.

Further Discussion of FORM

We would like now to discuss some of the aspects related to the FORM routine in greater detail. We note that of the eleven methods available in stage 2 of SIN, eight possess local clues which immediately identify them to FORM. Method 2, substitution for an integer power,

possesses clues which allow FORM to reject the method in some cases. Methods 8 (Rational) and 11 (Expansion) do not currently possess local clues in FORM and are attempted whenever FORM fails to find an applicable method.

As may be recalled from Chapter 2, one of the advantages of hypothesis formation is that one can attempt to fit the problem to the method at hand. Since FORM is quite aware of the methods which are available to it, some such "fitting" could be attempted. This was done in the case of algebraic integrands. If an expression is of the form $\sqrt{R(x)}$, where R is rational in x, then FORM will attempt to see if methods 3, 4, or 5 are applicable. If they are not, then this problem is going to cause some difficulty since it would appear that nothing else except stage 3 methods will be available to solve the problem. On the other hand it is possible that Methods 3, 4, or 5 are applicable, but that SCHATCHEN was unable to make the match. Two excuses can be made for SCHATCHEN in this event. One is that SCHATCHEN failed because the rational function R(x) was not expanded (e.g., $\sqrt{1 + x(1 - x)}$), or that the rational function was not completely rationalized (e.g., $\sqrt{x + \frac{x + 1}{x}}$). FORM will thus determine if these two transformations are applicable to R (not the whole integrand). If they are, the problem is transformed to account for these changes and an attempt will be made to consider Methods 3, 4, and 5 again. Hypothesis formation is thus shown to be able to localize the difficulty in a problem.

An Example of SIN's Performance

We shall now consider in some detail how SIN performs on the problem

$$\int \frac{\sqrt{A^2 + B^2 \sin^2 x}}{\sin x} \, dx$$

This problem stretches the capabilities of SIN a good deal. Thus it can be used to indicate some of the strengths and particularly the weaknesses in the program as it now stands. Our description will concentrate on the role that FORM plays in obtaining a solution.

This problem is not a simple one. So it will pass to stage 2, where FORM will examine it. It turns out that FORM will arrive at the same hypothesis regardless of whether it examines the numerator or denominator first, but it will be more instructive to see how it operates on the numerator. First, FORM will note the square-root (more precisely, the exponent of $\frac{1}{2}$). Since the base is not rational, which would indicate that Methods 3, 4, or 5 might be applicable, the root is ignored and attention is focused on the base $A^2 + B^2 \sin^2 x$. In this sum, the constant term A^2 is encountered, and it yields no clue. The factor B^2 is likewise a constant and yields no clue. This leaves the factor $\sin^2 x$. The exponent of 2 is not interesting. However, the base sin(x) does yield a clue since it is a trigonometric function with a linear argument. FORM will, therefore, call Method 6 in order to test the hpyothesis that the expression is an elementary function of trigonometric functions of x. Method 6 determines that the hypothesis is valid and will call SIN after making the substitution $y = \cos x$. The subproblem thus generated for SIN is

$$\int \frac{\sqrt{A^2 + B^2(1 - y^2)}}{1 - y^2} \, dy$$

As before, this is not a simple problem and again FORM is called in order to generate an hypothesis. Interest will quickly focus on the square-

root in the numerator. Though the base is a rational function, none of the clues in FORM appear to apply. As described in the discussion above, FORM will attempt to determine whether an expansion of the base is possible. Expansion is, of course, possible and yields the base $A^2 + B^2 - B^2y^2$ which matches the pattern used as a clue for Method 5. Method 5 is now called in order to determine whether an arctrigonometric substitution is possible in the revised problem which is

$$\int \frac{-\sqrt{A^2 + B^2 - B^2y^2}}{1 - y^2}\, dz .$$

Method 5 first validates the hypothesis. In order to determine which substitution is appropriate, the routine decides that $A^2 + B^2$ is positive and that $-B^2$ is negative in the manner described in the discussion of this method above. Method 5 will now make the substitution

$$z = \arcsin \frac{By}{\sqrt{A^2 + B^2}}$$

which is followed by a call to SIN with the subproblem

$$\int -\frac{1}{B} \frac{(A^2 + B^2)\cos^2 z}{1 - \frac{A^2 + B^2}{B^2}\sin^2 z}\, dz .$$

Once again the subproblem is not simple and FORM is asked to examine it. In the integrand only two factors are interesting, $\cos^2 z$ and $(1 - \frac{A^2 + B^2}{B^2}\sin^2 z)^{-1}$. Whichever FORM will be asked to examine first, the conclusion will be the same--a hypothesis that the integrand is an elementary function of trigonometric functions.

Method 6 will verify the hypothesis that only trigonometric functions are present and will make the substitution $w = \tan(z)$. This will result in yet another call to SIN with the subproblem

$$\int -\frac{1}{B} \frac{\frac{A^2 + B^2}{(1 + w^2)^2}}{(1 - \frac{A^2 + B^2}{B^2} \frac{w^2}{1 + w^2})}\, dw$$

This is a rational function and FORM will find no clue in this case. Since FORM also did not find any clue to reject the possibility that Method 2 (substitution for an integer power) is applicable, that method is called next. Method 2 cannot make a substitution, but will call Method 8 (rational) to solve this problem.

The rational function package will obtain this subproblem through the chaining process described above under Method 8. First, it will transform it by rationalization into a problem of the form given below

$$\int \frac{-B(A^2 + B^2)}{(1 + w^2)(B^2 - A^2 w^2)} dw$$

Then factorization and partial fraction decomposition will result in

$$\int \left[-\frac{B}{1 + w^2} + \frac{1}{2}A \frac{1}{Aw - B} - \frac{1}{2}A \frac{1}{Aw + B} \right] dw$$

Straight forward integration will now yield the integral

$$-B \arctan w + \frac{1}{2}A \log_e(Aw - B) - \frac{1}{2}A \log_e(Aw + B)$$

This result will be sent back to SIN for the arduous backward substitution: The first substitution is $w = \tan z$ which yields

$$-Bz + \frac{1}{2}A \log_e(A \tan z - B) - \frac{1}{2}A \log_e(A \tan z + B)$$

The second substitution is $z = \arcsin \frac{B}{\sqrt{A^2 + B^2}}\, y$. This results in

$$- B \arcsin \frac{B}{\sqrt{A^2 + B^2}} y + \frac{1}{2}A \log_e \left(\frac{A \frac{By}{\sqrt{A^2 + B^2}}}{\sqrt{1 - \frac{B^2}{A^2 + B^2} y^2}} - B \right)$$

$$- \frac{1}{2}A \log_e \left(\frac{A \frac{B}{\sqrt{A^2 + B^2}} y}{\sqrt{1 - \frac{B^2}{A^2 + B^2} y^2}} + B \right)$$

Note that tan arcsin C is transformed into $\frac{C}{\sqrt{1 - C^2}}$

The final substitution is y = cos x; this in turn yields

$$-B \arcsin\left(\frac{B}{\sqrt{A^2 + B^2}} \cos x\right) + \frac{1}{2}A \log_e\left(\frac{A\frac{B\cos x}{\sqrt{A^2 + B^2}}}{\sqrt{1 - \frac{B^2}{A^2 + B^2\cos^2 x}}} - B\right)$$

$$- \frac{1}{2}A \log_e\left(\frac{A\frac{B\cos x}{\sqrt{A^2 + B^2}}}{\sqrt{1 - \frac{B^2}{A^2 + B^2\cos^2 x}}} + B\right)$$

This is the result that SIN returns for the original problem. SIN does not simplify its results by rationalizing them or by combining logarithmic terms. This is certainly a drawback in this problem. Such simplifying transformations would result in the answer

$$-B \arcsin \frac{B}{\sqrt{A^2 + B^2}} \cos x - \frac{1}{2}A \log_e\left(\frac{A \cos x + \sqrt{A^2 + B^2 \sin^2 x}}{A \cos x - \sqrt{A^2 + B^2 \sin^2 x}}\right)$$

This result is to be compared with the answer in the table (Petit Bois, p. 138). That result is

$$B \arccos\left(\frac{B}{\sqrt{A^2 + B^2}} \cos x\right) - A \log_e(A \cot x + \sqrt{A^2 \csc^2 x + B})$$

In more familiar terms, the table's answer is

$$-B \arcsin\left(\frac{B}{\sqrt{A^2 + B^2}} \cos x\right) - A \log_e\left(\frac{A \cos x + \sqrt{A^2 + B^2 \sin^2 x}}{\sin x}\right)$$

This answer differs by a constant from the answer derived by SIN.

Although we can only guess at the method that the table's compiler used, we can arrive at some conclusions regarding weaknesses in SIN's method of solution.

Let us consider the first subproblem after the modification made to it by FORM.

$$\int \frac{-\sqrt{A^2 + B^2 - B^2y^2}}{1 - y^2} dy$$

Rewrite this as

$$\int \frac{(A^2 + B^2 - B^2y^2)}{(1 - y^2)} \frac{1}{\sqrt{A^2 + B^2 - B^2y^2}}\, dy$$

The transformation made above is a standard one in dealing with algebraic integrands. The integral above, after division, becomes

$$\int \left(B^2 - \frac{A^2}{(1 - y^2)}\right) \frac{1}{\sqrt{A^2 + B^2 - B^2y^2}}\, dy$$

Multiplying through we obtain two subproblems which together are simpler to solve than the combined problem. SIN, by not bringing the square-root to the denominator, unnecessarily complicates the work of the rational function package. This is certainly one of its weaknesses in dealing with algebraic integrands.

SAINT and SIN solutions of the same problem

As a further comparison of SAINT and SIN, we shall indicate how both operate on the problem

$$\int \frac{x^4}{(1 - x^2)^{5/2}}\, dx$$

This problem was chosen because it is discussed extensively in Slagle's thesis.

In SIN, after determining that the problem is not simple, the factor $(1 - x^2)^{-(5/2)}$ acts as a clue in FORM and generates a call to Method 5 which validates the hypothesis that an arctrigonometric substitution is possible. This method generates the subproblem

$$\int \frac{\sin^4 y}{\cos^4 y} dy$$

after making the substitution y = arcsin x.

Again, this is not a simple problem and this time sin(y) will act

as a clue for the hypothesis that only trigonometric functions are present. Method 6 validates this hypothesis and generates the subproblem

$$\int \frac{z^4}{1 + z^2} dx$$

after making the substitution $z = \tan y$.

This subproblem is rational and FORM finds no local clue. Method 2 is called and is ineffective. Method 8 (rational) is called and the rational function package returns the expression

$$\frac{z^3}{3} - z + \arctan z$$

as the integral.

Backward substitution yields

$$\frac{\tan^3 y}{3} - \tan y + y$$

and finally we obtain the integral

$$\frac{1}{3}\left(\frac{1 - x^2}{x^2}\right)^{-3/2} - \left(\frac{1 - x^2}{x^2}\right)^{-1/2} + \arcsin x$$

In SAINT, the solution of

$$\int \frac{x^4}{(1 - x^2)^{5/2}} dx$$

proceeds roughly as follows.

In this problem $y = \arcsin x$ is substituted yielding

I) $\int \frac{\sin^4 y}{\cos^4 y} dy$

as in SIN.

Subproblem I is transformed into

II) $\int \tan^4 y \, dy$

and into

III) $\int \cot^4 y \, dy$

both of which will now be added to the subproblem tree. Finally, $z = \tan \frac{1}{2}y$

transforms subproblem I into

IV) $\int 32 \frac{z^4}{(1 + z^2)(1 - z^2)^4} dz$

which is transformed by IMSLN into

V) $32 \int \frac{z^4}{(1 + z^2)(1 - z^2)^4} dz$

No more transformations are possible on subproblem I, so transformation will be attempted on subproblems II, III, and V.

Subproblem II is transformed by $z = \tan y$ into

VI) $\int \frac{z^4}{1 + z^2} dz$

IMSLN then performs the polynomial division and obtains

VII) $\int (-1 + z^2 + \frac{1}{1 + z^2}) dz$

From VII we obtain

VIII) $\int -dz,$

IX) $\int z^2 dz,$ and

X) $\int \frac{1}{1 + z^2} dz$

Subproblems VIII and IX are solved by the table look up in IMSLN. This leaves II, III, V and X.

III can be transformed by $z = \cot y$, into

XI) $\int \frac{-1}{z^4(1 + z^2)} dz$

and IMSLN will convert it to

XII) $-\int \frac{1}{z^4(1 + z^2)} dz$

By now only subproblems V, X, and XII remain to be considered. The transformation $w = \arctan z$ on subproblem X yields

XIII) $\int dw$

which IMSLN solves by the table look up. Now IMSLN realizes that subproblem VII has been completely solved and by backward substitution can obtain the final result

$$\frac{1}{3}\tan^3 \arcsin x - \tan \arcsin x + \arcsin x$$

We should note in the solution methods how SAINT keeps several options to the particular path to be followed in obtaining the answer. This is particularly noticeable in subproblem I which generates II, III, and IV. Only one of those three subproblems need be solved. SIN will generate only one subproblem, and will commit itself to using it. Of these subproblems only IV can truly be faulted. The $\tan \frac{1}{2} x$ transformation is generally to be eschewed if any other transformation is possible. However, the lack of communication between SAINT's heuristics make such a principle difficult to implement.

Furthermore, it appears that subproblem XIII should logically follow X. However, the cost of obtaining the character of subproblem X in SAINT forced the particular order of events to be followed. A mechanism like FORM would have simplified this situation tremendously.

CHAPTER 5

THE EDGE HEURISTIC

In this chapter we present the concepts underlying the Edge heuristic. The heuristic guesses the form of the integral and then attempts to obtain values for undetermined coefficients in that form. A program called Edge, which implements some of the ideas behind the Edge heuristic is described. The theoretical results related to this approach to integration are discussed.

Let us suppose that we are given an integrand which is in the form of a product. Then we can usually determine quite easily which factor in the product is a singular or outstanding factor in the sense that it is not contained in the other factors or their derivatives, nor can it be derived from the other factors or their derivatives through rational operations. In xe^{x^2}, the factor e^{x^2} is outstanding since x is contained in the derivative of this factor. The outstanding factor in $x^3\sqrt{1-x^2}$ is the factor $\sqrt{1-x^2}$. However, there may be several such outstanding factors as in $\sin x e^x$ where both $\sin x$ and e^x are not derivable from one another. In such a case we shall say that the first factor in a right to left scan of the expression is the outstanding factor. Moreover, in cases of functions such as $\sin(x)\cos(x)$ no factor is outstanding. Here we shall choose the first factor on the right.

Given that we have decided on an outstanding factor in the integrand, we can frequently make an educated guess regarding the form of the integral, assuming, of course, that the integral can be expressed in finite terms.

Suppose the integral $f(x)$ has an outstanding factor of the form $e^{g(x)}$, say, $f(x) = h(x)e^{g(x)}$ then we can guess that $\int f(x)dx$ is of the form

$$a(x)e^{g(x)} + b(x) = \int f(x)dx = \int h(x)e^{g(x)}dx$$

where $a(x)$, $b(x)$ are undetermined functions of x, and where $a(x)$ will not involve $e^{g(x)}$.

Certainly $\int f(x)dx$ must contain $e^{g(x)}$ since one cannot otherwise obtain such a function through differentiation. If $\int f(x)dx$ has a nonlinear occurrence of $e^{g(x)}$ then so will its derivative, but this nonlinear occurrence will not cancel in $f(x)$.

Given the above choice for $\int f(x)dx$, then by differentiation we obtain

$$a(x)e^{g(x)}g'(x) + a'(x)e^{g(x)} + b'(x) = f(x) = e^{g(x)}h(x)$$

A simple choice for the value of $a(x)$ can be obtained by requiring that the first coefficient of $e^{g(x)}$ on the left be equal to the coefficient of $e^{g(x)}$ in f. Using this choice we obtain

$$a(x) = \frac{f(x)}{e^{g(x)}g'(x)} = \frac{h(x)}{g'(x)}$$

The value of b(x) is obtained in a subproblem.

$$b(x) = \int -a'(x)e^{g(x)}dx$$

Hopefully, the choice of a(x) made above will yield a simpler integration problem for the determination of b(x) than the original problem. Let us consider a simple example using this guessing procedure.

$$f(x) = xe^x$$

$$a(x)e^x + b(x) = \int f(x)dx$$

$$a(x)e^x + a'(x)e^x + b'(x) = xe^x$$

$$a(x) = \frac{xe^x}{e^x} = x$$

$$a'(x) = 1$$

$$b(x) = \int -1 \cdot e^x dx = \int -e^x dx$$

The subproblem for b(x) is certainly simpler than the original problem. It will be instructive to consider how the method outlined above will handle such a problem. Below we shall usually ignore the functional characterization of a(x) and b(x).

$$b = \int -e^x dx$$

$$a_1 e^x + b_1 = b$$

$$a_1 e^x + a_1' e^x + b_1' = b' = -e^x$$

$$a_1 = \frac{-e^x}{e^x} = -1$$

$$a_1' = 0$$

$$b_1' = \int -0 \cdot e^x dx = \int 0 \, dx = \text{constant}$$

$$b = a_1 e^x + b_1 = -e^x + \text{constant}$$

Finally,

$$\int f(x)dx = xe^x - e^x + \text{constant}$$

Let us now consider another example using this procedure.

$$f(x) = x \cos x^2 e^{\sin x^2}$$

The outstanding factor in $f(x)$ is $e^{\sin x^2}$

$$ae^{\sin x^2} + b = \int f(x)dx$$

$$ae^{\sin x^2} \cos x^2 \, 2x + a'e^{\sin x^2} + b' = x \cos x^2 e^{\sin x^2}$$

$$a = \tfrac{1}{2}$$

$$a' = 0$$

$$b' = 0, \; b = \text{constant}$$

$$\int f(x) \, dx = \tfrac{1}{2} e^{\sin x^2} + \text{constant}$$

The first of the two problems above is usually solved by Integration-by-parts. However, that method requires an integration step (i.e., $\int e^x dx$) which we did not perform. Furthermore, the integration by parts method is inapplicable in the second problem above. The latter problem is handled by the Derivative-divides method such as is used in SIN's first stage. So the analysis performed by the Edge heuristic and in particular the analysis of Edge that we have been presenting is different from either of these two general methods of integration.

An analysis which is similar, but more complex than the one made by Edge is employed by Method 7 of SIN's second stage. Let us consider the manner in which the method proceeds in light of the discussion above.

We recall that Method 7 deals with integrands of the form $R(x)e^{P(x)}$ where R is rational and P is a polynomial in x.

An example solved by this method is

$$f(x) = (2x^2+1)e^{x^2}$$

Edge would in this case guess

$$a(x)e^{x^2}+b(x) = \int f(x)dx$$

and

$$a(x) = \frac{2x^2+1}{(x^2)'} = \frac{2x^2+1}{2x}$$

Method 7 is superior in this case in that it considers the R(x) factor term by term. Thus, it would guess

$$a(x) = \frac{2x^2}{(x^2)'} = \frac{2x^2}{2x} = x$$

It turns out that this is the correct value for a(x) since the integral is exactly xe^{x^2}.

On a more complex problem such as

$$\frac{2x^6 + 5x^4 + x^3 + 4x^2 + 1}{(x^2+1)^2} e^{x^2}$$

Method 7 would proceed by first letting

$$a(x) = \frac{2x^6}{(x^2)'(x^2+1)^2} = \frac{x^5}{(x^2+1)^2}$$

The subproblem it generates is

$$\frac{4x^4 + x^3 + 5 - \frac{4}{x^2+1}}{(x+1)^2} e^{x^2}$$

Now it lets

$$a_1(x) = \frac{4x^4}{(x^2)'(x^2+1)^2} = \frac{2x^3}{(x^2+1)^2}, \text{ etc.}$$

Finally, the result is

$$\frac{x^5 + 2x^3 + \frac{1}{2}x^2 + x + \frac{1}{2}}{(x^2+1)^2} \; e^{x^2}$$

or

$$\frac{2x^3 + 2x + 1}{2(x^2+1)} \; e^{x^2}$$

Thus, we see that although the heuristic of guessing the form of the integral is correct in the two examples above, the particular mechanism for guessing the values of the undetermined coefficients which is employed in Edge is not sufficiently powerful. We shall now indicate two other difficulties with the analysis of Edge described above.

Let us recall that Method 1 of SIN's second stage handles integrands of the form Elem(e^x). This method substitutes $y=e^x$. In the case of rational functions of exponentials this substitution yields a rational function. Thus, for example,

$$f(x) = (e^x+1)e^{2x}$$

becomes

$$(y+1)y$$

after making the substitution. The rational function package will expand this integrand and integrate the resulting quadratic in y. Edge would guess the form of the integral without making a corresponding expansion. This leads to an incorrect guess of the form

since the two factors in f(x) are closely related. Had Edge expanded the integrand and integrated the terms separately, it would have easily obtained the integral of f(x).

Another difficulty with the manner in which Edge guesses the form of an integral is shown in

$$f(x) = \frac{1}{e^x+1} e^{-x}$$

Method 1 of SIN's second stage would yield a rational function which would be factored and expanded in partial fractions by the rational function package. Here again the two factors f(x) are closely related and thus the guess of the form of the integral made by Edge and the resulting guesses of the coefficients will fail to yield the integral. A partial fraction expansion is required if the integrand is a rational function of related terms.

While keeping these weaknesses of Edge in mind, we shall continue to consider how the guessing heuristic operates on outstanding factors of different forms.

Let us suppose that

$$f(x) = h(x) \log(g(x))$$

and that the logarithmic factor is the outstanding factor in f(x). A good guess of the form $\int f(x)dx$, if it exists, is

$$c\log^2(g(x)) + a(x)\log(g(x)) + b(x) = \int f(x)dx$$

where c is a constant and a(x) does not involve log(g(x)).

The $\log^2$ term is necessary (e.g., $f(x) = 1/x \log x$), but its coefficient is only a constant. Otherwise the derivative of the from above would contain a $\log^2$ term which would not cancel in $f(x)$.

Differentiating we obtain

$$2c\frac{g'(x)}{g(x)} \log g(x) + a\frac{g'(x)}{g(x)} + a' \log g(x) + b' = h(x) \log g(x)$$

or

$$(2c\frac{g'(x)}{g(x)} + a')\log g(x) + a\frac{g'(x)}{g(x)} + b' = h(x) \log g(x)$$

In the above we grouped the terms involving the outstanding factor $\log g(x)$. We note two differences from the exponential case. First there is the constant c which did not arise before. Then the coefficient of the log term is a' instead of a. We can solve for $a(x)$ by using the relationship

$$a' = h(x) - 2c \frac{g'(x)}{g(x)}$$

$$a = \int h(x)dx - 2c \log g(x)$$

We now use the fact that $a(x)$ is independent of $\log g(x)$ in order to obtain a value for c. That is, if $\int h(x)dx$ has a term involving $\log g(x)$, the c is chosen so as to cancel that term. Otherwise, we chose c=0. The value of b' is determined by the relationship.

$$b' = -a\frac{g'(x)}{g(x)}$$

Let us consider an example.

$$f(x) = (x + 1/x)\log g(x)$$

$$c \log^2 x + a \log x + b = \int (x+1/x)\log x \, dx$$

$$(2c/x + a')\log x + a/x + b' = (x + 1/x)\log x$$

$$a = \int (x + 1/x)dx - 2c \log x = 1/2\, x^2 + \log x - 2c \log x$$

$$2c = 1,\ c = 1/2,\ a = 1/2\, x^2$$

$$b' = -a/x = -1/2\, x$$

$$b = -1/4\, x^2$$

$$\int (x + 1/x)\log x \, ex = 1/2 \log^2 x + 1/2\, x^2 \log x - 1/4\, x^2$$

It should be noted that $\int (x + 1/x)dx$ can, of course, also be obtained by a guess of the integral.

The guess for the logarithmic case generalizes when $f(x)$ is of the form

$$f(x) = h(x) \log^n g(x), \quad n > 0$$

In this case we can guess

$$c \log^{n+1} g(x) + a \log^n g(x) + b = \int h(x) \log^n g(x) dx$$

with a,b,c determined using the same method as above.

Let us consider how we can capitalize on our experience of the types of outstanding factors dealt with above. Suppose f(x) is of the form

$$f(x) = \frac{h(x)}{1 + g^2(x)}$$, where $\frac{1}{1 + g^2(x)}$ is the outstanding factor.

The argument now proceeds as follows: One could arrive at a factor $\frac{1}{1 + g^2(x)}$ by two routes which do not involve complex constants:

a) $\log(1 + g^2(x))$

b) $\arctan g(x)$.

In either case the coefficients must be constants since if they were not the derivatives would contain terms more complex than found in the integrand. Thus the guess is

$$c \log(1 + g^2(x)) + d \arctan g(x) = \int f(x)dx$$

$$\frac{2cgg'}{1 + g^2} + \frac{dg'}{1 + g^2} = \frac{h(x)}{1 + g^2}$$

$(2gc + d)\, g' = h(x)$ where c, d are constants.

Consider $f(x) = \frac{x}{1 + x^4}$

$(2x^2c + d)2x = x$

$2x^2c + d = \frac{1}{2}$

$c = 0,\ d = \frac{1}{2}$

$\int f(x)dx = \frac{1}{2}\arctan x^2$

We should note that our guess fails in such cases as $\frac{x^5}{1 + x^4}$ in which division must be attempted first, or in the case of $\frac{1}{1 + \tan^2 x}$ which is equivalent to $\cos^2 x$.

In order to contrast the Edge heuristic approach with that used in Stage 2 of SIN, let us consider functions of the form

$$f(x) = \frac{h(x)}{(1 - g^2(x))^{n/2}} \text{ , n a positive integer}$$

An educated guess for the form of the integral of f(x) is

$$\frac{a}{(1 - g^2(x))^{n/2 - 1}} + b = \int f(x)dx, \text{ unless } n = +1$$

If n = +1, then we shall also consider the possibility of a c arcsin(g(x)) term, where c is a constant.

An example we considered in Chapter 4 is

$$f(x) = \frac{x^4}{(1 - x^2)^{5/2}}$$

$$\frac{a}{(1 - x^2)^{3/2}} + b = \int \frac{x^4}{(1 - x^2)^{5/2}}\, dx$$

$$\frac{a(\frac{-3}{2})(-2x)}{(1 - x^2)^{5/2}} + \frac{a'}{(1 - x^2)^{3/2}} + b' = \frac{x^4}{(1 - x^2)^{5/2}}$$

$$a = \frac{x^4}{3x} = \frac{x^3}{3}$$

$$a' = x^2$$

$$b' = \frac{-x^2}{(1 - x^2)^{3/2}}$$

Now we shall generate a subproblem.

$$\frac{a_1}{(1 - x^2)^{1/2}} + b_1 = \int \frac{-x^2}{(1 - x^2)^{3/2}}\, dx$$

$$\frac{a_1(-1/2)(-2x)}{(1 - x^2)^{3/2}} + \frac{a_1'}{(1 - x^2)^{1/2}} + b_1' = \frac{-x^2}{(1 - x^2)^{3/2}}$$

$$a_1 = \frac{-x^2}{x} = -x$$

$$a_1' = -1$$

$$b_1' = \frac{1}{(1 - x^2)^{1/2}}$$

In this case we shall guess

$$a_2(1 - x^2)^{1/2} + c \arcsin x = \int (1 - x^2)^{-1/2} dx$$

$$\frac{a_2(1/2)(-2x)}{(1 - x^2)^{1/2}} + \frac{c}{(1 - x^2)^{1/2}} = \frac{1}{(1 - x^2)^{1/2}}$$

$$-xa_2 + c = 1$$

$$c = 1$$

$$a_2 = 0$$

The final result is

$$\int \frac{x^4}{(1 - x^2)^{5/2}} dx = \frac{x^3}{3} (1 - x^2)^{-3/2} - x(1 - x^2)^{-1/2} + \arcsin x$$

We should like to mention how Edge handles trigonometric functions. For outstanding factors of the form sin(g(x)) it guesses cos(g(x)) and it guesses cos(g(x)) for outstanding factors of the form sin(g(x)). However, this manner of dealing with trigonometric functions is not necessarily the best one. Edge should in some cases consider the complex exponential form of the trigonometric functions. In this way $\int \sin^n x \, dx$ can be found easily for integral values of n after expanding the complex exponential form of the integrand. By keeping the trigonometric form Edge is forced to deal with methods such as "solution by transposition" which occurs in $\int \sin x \, e^x dx$ when one of the subproblems is $\int -\sin x \, e^x dx$.

We have indicated above some examples in which Edge fails to

make a good guess for the form of the integral or the values of the undetermined coefficients in the form. Thus, it is necessary to determine whether Edge is progressing toward a solution. If the outstanding term involves an exponent and the absolute value of the exponent is decreasing, the routine thinks that it is making progress. The same is true if another factor in the integrand is exponentiated and its exponent is decreasing while the outstanding factor remains the same. The program is certainly not progressing if it obtains a subproblem which is exactly the same as some previous subproblem, though a solution by transposition is attempted if a subproblem is a constant multiple other than one of some previous subproblem. In the above we have indicated some cased in which the form has coefficients which were constrained to be constants. The current version of Edge handles these cases by attempting a guess which ignores a term (usually the one with a constant multiple). If that guess fails to yield the integral using the progress information outlined above, the program backs up and introduces a new term in the form while eliminating another term. In this manner Edge performs a depth first search.

Below we would like to indicate the theoretical results which underlie the Edge heuristic.

Historically, the quest for results regarding the form of an integral goes back to the early nineteenth century. Laplace conjectured that the integral of an algebraic function (y is algebraic

in x if $P(x, y) = 0$ where P is a polynomial with constant coefficients) need contain only those algebraic functions which are present in the integrand. This conjecture was proved by Abel. Liouville examined the form of the integral of an elementary function in a series of papers in the 1830's. Before we present the statement of Liouville's main theorem, we shall need some preliminary considerations. An important feature of Liouville's theory of integration is a hierarchy of elementary functions. In level 0 of this hierarchy are the algebraic functions. The monomial of level 0 is x. A monomial of level i + 1 is a function represented by e^y or log y, where y is a function of level i and where the monomial has no representation which is of lower level than i + 1. Level i + 1 also contains all functions which are algebraic combinations of monomials of level i + 1 with functions of lower levels provided again that those functions have no representation of lower level. Thus, xe^{x^2} is of level 1 and $e^x e^{e^x} + \log(1 - ix^2)$ is of level 2. We should note that this hierarchy includes all trigonometric and arctrigonometric functions by using their complex exponential and logarithmic forms in order to classify them.

Given a representation of an elementary function one can list the monomials and algebraic functions of these monomials which were combined to form the function. Among the monomials and the algebraic functions there will be some which are of the highest level. Choose one such function and call it the <u>principal function</u>. Thus, the

original function is a rational combination of the principal functions with functions of equal or lower level. The principal function in xe^{x^2} is e^{x^2} and the principal function in $\frac{e^x+1}{e^{2x}+3e^x}$ is e^x. It is the concept of a principal function which we were striving for when we defined the concept of an outstanding factor in an integrand. We noted above some of the difficulties that one encounters in making an educated guess for the form of the integral when using only the notion of an outstanding factor. The principal function concept surmounts these difficulties.

We are now in a position to ask whether there are any more monomials and algebraic functions in the integral of a function than in the function itself. The answer provided by Liouville's general theorem is that except for logarithmic extensions there are none. Liouville's theorem states that

$$\int f(x)dx = v_o(x) + \sum_{i=1}^{n} c_i \log v_i$$

where the ci's are complex constants and the v_i are rational functions in the monomials and algebraic functions of these which appear in f [54].

Liouville's theorem itself gives a strong rationale to the Edge heuristic since it makes strong restrictions on the possible forms

of the integral. Recently, and independently of our work on Edge, Risch [53] has strengthened the Liouville theorem by showing that the constants c_i need only be algebraic over the field of constants generated by the constants in f(x) with the ground field of the rational numbers. Risch has also given a decision procedure for those functions obtained without using any algebraic operations other than rational operations. His method is similar to the one employed in Edge in that it relies on knowing the possible form of the integral. However, it is superior to Edge in the manner in which it obtains the undetermined coefficients and in its use of partial fraction decomposition with respect to the principal function in the integrand. When algebraic operations are allowed in the integral, Risch believes that the integration problem may in general be recursively unsolvable. (See Appendix B where the integration problem is shown to be unsolvable using a different formulation than Risch's.) However, he is optimistic about integrands which are algebraic functions of level 0 in our hierarchy.

We believe that methods which rely on guessing the form of the integral such as Edge or ones based on Risch's algorithm will in the near future provide us with very powerful integration programs. However, the amount of machinery that they call into play and their use of radical transformations such as the complex exponential form of the trigonometric functions indicate that those methods are not to be applied when more specific and presumably more efficient methods are available.

Chapter 6

SOLUTION OF ORDINARY DIFFERENTIAL EQUATIONS

As a first approximation one might attempt to treat the problem of solving ordinary differential equations by using a similar strategy to the one used in SIN for integration problems. Let us recall that SIN used a three stage approach. First it attempted to solve the problem using simple methods. Next the FORM routine attempted to use local clues to determine which one of a specific set of methods was applicable to the problem. Finally the Edge routine employed a more general method of solution. In this chapter we shall consider how such a strategy would fare in the problem domain of first order, first degree ordinary differential equations (i.e. $P(x,y)y'+Q(x,y)=0$). We shall indicate the approach that was finally taken and describe the methods of solution which were programmed.

There appears to be general agreement in the texts of ordinary differential equations regarding the elementary forms of differential equations. Linear, exact and separable equations seem to constitute the universal choice as elementary forms. They are, respectively, of the form $f(x)y'+g(x)y+h(x)=0$, $P(x,y)dx+Q(x,y)dy=0$, where $\frac{\partial P}{\partial y}=\frac{\partial Q}{\partial x}$, and $A(x)B(y)dx+C(x)D(y)dy=0$. These forms are relatively easy to recognize, and immediately reduce to integration problems. We shall adopt the usual convention that a reduction of a differential equation to one or more integration problems constitutes a solution of the equation even if the expressions to be integrated cannot be integrated in finite terms. Functions which can be ex-

pressed in terms of elementary functions and integrals of elementary function are called Liouville functions. Due to the above-stated properties of linear, exact, and separable equations, the set of methods which determine whether the equation matches one of the forms constitute a reasonable analogue to SIN's first stage.

When we consider finding an analogue to the FORM routine of SIN, we immediately arrive at difficulities. It is rare that one can make a slight change to a differential equation and still be able to use the same method of solution, let alone obtain a similar solution. Let us consider how the method of solution changes as we modify the five equations below. The methods of solution used (i.e., linear, exact, homogeneous, Bernoulli, and linear coefficients) will be described later.

1) $2xy' + y+x+1=0$

linear

2) $2xy'+y(y+x+1)=0$

Bernoulli

3) $(2x+y)y'+y+x+1=0$

linear coefficients

4) $x(x+y)y'+y(y+2x)=0$

homogeneous

5) $x(x+2y)y'+y(y+2x)+1=0$

exact

It should be noted that none of the methods mentioned above is applicable to any of the other four problems. The situation is

even more serious when we note that equation 6 is not integrable in terms of Liouville functions, but equation 7, which varies from equation 6 by only the addition of the constant 1, does possess a Liouville solution (see Ritt [54] p. 73).

6) $x^2y'+x^2(y^2-1)-2=0$

7) $x^2y'+x^2(y^2-1)-1=0$

Since the equations above appear quite similar, any test based on local clues only is going to fare quite badly. Thus the possibility of implementing an analogue to SIN's FORM routine does not appear very promising. One could of course, use global clues (such as the number of occurrences of x and y in the coefficient of y') to conclude that certain methods are inapplicable (for example, the linear method is inapplicable if there are any occurrences of y in the coefficient of y'). However, this approach is not likely to give us a great increase in efficiency.

On the basis of the difficulty just noted, one would suppose that a practical general method for solving first order, first degree ordinary differential equations is not likely to exist. Surprisingly, a general method does exist. It is known as the multiplier method. It can be shown that if a Liouville solution exists, then there also exists a Liouville function u(x,y), which can be used to multiply both sides of the equation and obtain an exact differential equation and thus an immediate solution. That is, given $P(x,y)dx+Q(x,y)dy=0$, then $uPdx+uQdy=0$ satisfies $\frac{\partial}{\partial y}(uP)=\frac{\partial}{\partial x}(uQ)$.

There is, however, a slight catch in the multiplier method - it is very hard to find an appropriate multiplier except in special

cases. In fact, several texts caution their readers against trying to consider finding multipliers to differential equations. The Liouville theory (see Chapter 5) yields a form that an elementary solution to a first order differential equation must satisfy. However it does not appear likely that one could write a method like Edge which would exploit this information, except in special cases. Negative results such as those in Appendix B appear to dampen the hope that one could find a general method for solving differential equations.

We thus conclude that finding an analogue to SIN's strategy in the domain of differential equations is quite difficult if not impossible. We can, however, decrease our expectations and follow the traditional technique given in texts on differential equations. That is we can determine if the problem is solvable by one of a set of special methods by examining the applicability of the methods one at a time. It is this approach which was implemented. We were reduced to a search for a method because of our inability to either localize the problem or to find a simple model for it. The crucial role of constants in determining a solution frustrates even the most primitive simplifying considerations. There is one consolation in the approach taken, and that is that once we find a method which is applicable it is either immediately reducible to integration problems or reduces to simple problems (i.e., linear, exact, or separable) in one or at most two steps. Furthermore, these steps are known in advance in most cases.

Eight methods of solution for first order, first degree differential equations were coded. These include most of the

methods for solving first order equations taught in an introductory course on ordinary differential equations. As stated above, the methods are examined in turn in order to determine if they are applicable. The simple methods are attempted first. These will all call SIN whenever they apply in order to solve some integration problems. The five other methods will generate subproblems which are usually either linear, exact or separable.

The conventions for stating the problem to the machine are the ones used in the text books or the tables. When the dependent variable is x, and the independent variable is y, the problem may be stated in either form I or II:

I $P(x,y)y'+Q(x,y)$

II $P(x,y)dx+Q(x,y)dy$

It is assumed that the expression given is to be equated to 0. The result, if found, will be stated in the form

$$f(x,y)=Co \quad ,$$

where Co is a constant of integration. As will be seen, no attempt is currently made to solve for y or to perform other simplifications such as eliminating logs in the resulting expression.

Top level control resides in a routine called SOLDIER (SOLution of DIfferential Equation Routine). SOLDIER will translate the problem statement into the form (either I or II) desired by the particular method. It will be noted that books tend to state a problem applicable to a given method in only one of the two forms (e.g., linear equations are usually in form I, and exact in form II). No attempt was made to use this fact as a clue to a solution.

We now shall proceed in describing the methods.

Method 1 LINEAR

FORM $f(x)y'+g(x)y+h(x)=0$

Procedure

Let $P(x)=\frac{g(x)}{f(x)}$, $Q(x)=\frac{h(x)}{f(x)}$

The solution is

$$ye^{\int P dx} + \int Q \left[e^{\int P(x)dx}\right] dx = Co$$

Notes

The recognition of this form is done by a SCHATCHEN pattern. Since equations of the form $f(x)y'+g(x)[h(x)y+k(x)]=0$ will not be recognized as linear by SCHATCHEN using the pattern given above, expansion is attempted as a heuristic aid to recognizing forms. Expansion is, however, attempted only when a single occurrence of y appears in the equation. Thus $f(x)y'+g(x)y+h(x)[y+k(x)]=0$ is not expanded and is not recognized as a linear differential equation.

Examples

1) $y'+y+x=0$

becomes

$$ye^{x}+\int xe^{x}dx=Co$$

Thus solution is

$$ye^{x}+xe^{x}-e^{x}=Co$$

2) $xy'+xy+1=0$

results in $ye^{x}+\int \frac{e^{x}}{x} dx = Co$

Method 2 SEPARABLE

FORM $A(x)B(y)dx+C(x)D(y)dy=0$

Procedure The solution is

$$\int \frac{A(x)}{C(x)}\,dx + \int \frac{D(y)}{B(y)}\,dy = Co$$

Notes

No attempt is made to recognize this form except through SCHATCHEN's matching techniques. Thus no factorization of the equations is attempted. That is the factorization must be explicit although several factors may involve just y or just x.

Examples

1) $x(y^2-1)dx - y\,(x^2-1)dy=0$

becomes

$$\int \frac{x}{x^2-1}\,dx + \int \frac{-y}{y^2-1}\,dy=Co$$

Thus the solution is

$$1/2 \log (x^2-1) - 1/2 \log (y^2-1)=Co$$

This answer is normally simplified on tables to become $\frac{x^2-1}{y^2-1}=Co$ or $(x^2-1)=Co(y^2-1)$. As stated above no attempt is currently made to perform such simplifications.

2) $e^x \sin y\, y' + x\cos y=0$

becomes

$$\int \frac{\sin y}{\cos y}dy + \int xe^{-x}\,dx=Co$$

or

$$-\log \cos y - xe^{-x} - e^{-x} =Co$$

The transformation of this problem to the dx, dy form is performed by SOLDIER.

Method 3) Exact - Multipliers

Exact FORM $P(x,y)dx + Q(x,y)dy=0$

The method is applicable whenever

$$\frac{\partial P}{\partial y} = \frac{\partial Q}{\partial x}$$

The answer is

I $$\int P\,dx + \int \left[Q - \frac{\partial}{\partial y} \left[\int P\,dx \right] \right] dy = Co$$

Since this method is closely related in form requirements and solution method to certain special cases of the multiplier method, these cases are considered here.

a) If $\dfrac{\frac{\partial P}{\partial y} - \frac{\partial Q}{\partial x}}{Q} = h(x)$, i.e., the quotient is just a function of x, then the multiplier is $e^{\int h(x)dx}$

Procedure Let $\bar{P}(x,y) = P(x,y)*\text{multiplier}$, $\bar{Q}(x,y) = Q(x,y)*\text{multiplier}$

$\bar{P}$ and $\bar{Q}$ are guaranteed to satisfy

$$\frac{\partial \bar{P}}{\partial y} = \frac{\partial \bar{Q}}{\partial x}$$

The solution is obtained using the procedure of equation I above with P,Q replaced by P and Q, respectively.

b) If $\dfrac{\frac{\partial Q}{\partial x} - \frac{\partial P}{\partial y}}{P} = k(y)$, that is the quotient is a function of y only, then

$e^{\int k(y)dy}$ is a multiplier. Proceed as in step a).

c) If $\dfrac{\partial P}{\partial y} = -\dfrac{\partial Q}{\partial x}$ and $\dfrac{\partial P}{\partial x} = \dfrac{\partial Q}{\partial y}$

then the multiplier is $\dfrac{1}{P^2+Q^2}$. Proceed as in step a)

Notes

SCHATCHEN is used to perform the matching required in testing to determine if $\dfrac{\partial P}{\partial x}$ equals $\dfrac{\partial Q}{\partial y}$. Clearly a matching program such as Martin's [37] would be preferable in this case since no pattern matching is necessary, but only a match for equivalence.

The division steps employ only SCHVUOS's limited simplification methods for quotients. Thus no factorization is attempted. At present there exists no simplification program which can simplify quotients well. For example

$$\frac{e^{2x}+2e^{x}+1}{e^{x}+1}$$

is not simplified to $e^{x}+1$ by any reported simplification program.

Another approach to determining the applicability of the first three multiplier cases is to differentiate the quotient with respect to y in the first case and with respect to x in the second case. This reduces the recognition problem to a match for equivalence to 0. In this manner we avoid placing constraints on the simplification program for determining the applicability of the method. However this technique does not yield the desired value of the quotients.

There exist many other special cases for the multiplier. In fact the origin of Lie Groups was motivated by considerations regarding the families of differential equations which are solved by particular multipliers.

Examples

1) $(4x^3y-12x^2y^2+5x^2+3x)y'+6x^2y^2-8xy^3+10xy+3y=0$

Solution is

$$2x^3y^2-4x^2y^3+5x^2y+3xy=Co$$

2) $(2xy+5x+1)y'+y^2=0$

Solution is

$$xy^2e^{-5/y}+\int e^{-5/y}dy = Co$$

Method 4 Bernoulli

FORM $f(x)y'+g(x)y + h(x)y^n=0$, where n is a constant, $n \neq 1$

Procedure

Substitute $u(x)=y^{1-n}$ in order to obtain the linear equation

$$f(x)\, u'+(1-n)\, g(x)u+ (1-n)h(x)=0$$

Notes

The form of the equation is tested by SCHATCHEN. As in the linear case expansion will be attempted to aid the pattern match, but only when there are exactly two occurrences of y in the equation.

Examples

1) $x^2(x-1)y' = y^2 - x(x-2)y=0$

transformed into the linear equation

$$y' + \frac{(x-2)}{x\,(x-1)}\, y + \frac{1}{x^2\,(x-1)} =0$$

2) $3xy' - 3xy^4\log_e x - y = 0$

is transformed into

$$y' + \frac{y}{x} + 3 \log_e x = 0$$

Method 5 Homogeneous

FORM $P(x,y)dx + Q(x,y)dy = 0$

where P and Q are homogeneous functions in x and y of some degree, n, say.

Procedure The substitution $u(x) = \frac{y}{x}$ is made. After factoring x^n from the equation, one obtains an equation with the variables separable (Method 2).

Notes

This is a common form for a differential equation. It is

a subcase of method 8, but is given special treatment here because of the frequency and ease of recognition of this form.

The factorization of x^n from the equation must, in general, be performed in order to have the result recognized as separable. The recognition of homogeneity and factorization are performed by SCHATCHEN and SCHVUOS and thus are not unusually powerful. For example $\frac{x^2+xy}{x} y'+y=0$ is not recognized as homogeneous.

Examples

1) $3x^2y' - 7y^2 - 3xy-x^2=0$

solution is

$$\log_e x - \frac{3}{\sqrt{7}} \arctan \sqrt{7}\,\frac{y}{x} = Co$$

2) $2x(y^3+5x^2)\, y'+y^3-x^2y =0$

solution is

$$\log_e x + \frac{10}{9} \log_e \frac{y}{x} - \frac{2}{9} \log_e\left(3+\frac{y^2}{x^2}\right) = Co$$

Method 6 Almost Linear

FORM $f(x)g(y)\, y' + h(x,y) = 0$

where

$$h(x,y) = k(x)l(y)+m(x)$$

and

$$l'(y) = g(y)$$

Procedure

Substitute $u(x) = l(y)$ resulting in the linear equation

$$f(x)u' + k(x)\, u+m(x)=0$$

Notes

This is a method which is rarely indicated in the texts.

Examples

1) $xyy' + 2xy^2+1=0$

substitution is $u(x)=y^2$

yielding

$$\frac{1}{2}xu' + 2xu+1=0$$

2) $x^2\cos y\, y' + \sin y + e^{\frac{1}{x}} = 0$

substitution $u = \sin y$

yields

$$x^2u' + u + e^{\frac{1}{x}} = 0$$

Method 7 Linear coefficients.

FORM $y'+F\left(\frac{ax+by+c}{a'x+b'y+c'}\right) = 0$ where a,b,c,a',b',c' are constants and $ab' - a'b \neq 0$

Procedure

Substitute

$$x^* = x - \frac{b'c - bc'}{a'b - ab'}, \quad y^* = y' - \frac{ac' - a'c}{a'b - ab'}$$

and obtain a homogeneous problem (method 5).

Notes

Recognition is based on matching

$$A(ax+by+c)^n\ (a'x+b'y+c')^{-n}$$ repeatedly

in $F(x,y)$, where a,b,c,a',b',c' are assumed to remain fixed in $f(x,y)$.

Examples

1) $(4y+11x-11)y'-25y-8x+62=0$

answer is

$$\log_e\left(x - \frac{1}{9}\right) - \frac{1}{2}\log_e\left(1+2\left(\frac{y-\frac{22}{9}}{x-\frac{1}{9}}\right)\right)$$

$$+ 3/2\ \log_e\left(-4 + \frac{y-\frac{22}{9}}{x-\frac{1}{9}}\right) = c_0$$

2) $(y+x-1)\ y'-y+2x+3=0$

answer is

$$\log_e\left(x+\frac{2}{3}\right)+\frac{\sqrt{2}}{2}\ \arctan\frac{\sqrt{2}}{2}\left(\frac{y-\frac{5}{6}}{x+\frac{2}{3}}\right)$$

$$+\ 1/2\ \log_e\left(2+\left(\frac{y-\frac{5}{6}}{x+\frac{2}{3}}\right)^2\right)=Co$$

Method 8 Substitution for $x^n y$

FORM $y'+L(x,y)=0$

where $L(x,y)=\frac{y}{x}\,H\,(x^n y)$,

Here H is a function of a single argument,

and n is a constant to be determined.

Procedure Substitute $u(x)=x^n y$ resulting in the separable equation

$$\frac{du}{u(n-H(u))}=\frac{dx}{x}$$

The method employed to recognize this form uses the implicit function theorem to yield an equation in n.

Consider

$G(x,y)=\frac{x}{y}\,L(x,y)$

We wish to determine if $G(x,y)=H(x^n y)=H(u(x,y))$.

The implicit function theorem states that this relation will hold if and only if

$$\frac{\partial G}{\partial x}\frac{\partial u}{\partial y}-\frac{\partial G}{\partial y}\frac{\partial u}{\partial x}=0$$

Note that this equation represents the Jacobian in the two variable case. Since $u(x,y)=x^n y$, we obtain the following relationships:

$$\frac{\partial G}{\partial x} x^n - n\frac{\partial G}{\partial y} x^{n-1} y = 0$$

or

$$n = \frac{x\frac{\partial G}{\partial x}}{y\frac{\partial G}{\partial y}}$$

If n is known, we can determine whether the above relationships holds. However we can also use this relationship to generate a value for n. If the right hand side of the last equation is a constant than a substitution with n as that value is possible. If it is not a constant, the method is inapplicable.

Notes

This method is a generalization of the homogeneous case (Method 5). The method is rarely described although it accounts for many of the substitutions in the first 367 equations in Kamke [30]. In some of these cases Kamke prefers to give other methods of solution. For example, in (I 293) $x(y^2-3x)y'+2y^3-5xy=0$, Kamke suggests dividing by $x^{27} y^{16}$ instead of substituting $u(x,y) = x^{-1/2} y$.

In this method we resorted to a special purpose matching rule instead of using SCHATCHEN. The use of the implicit function theorem was suggested by Engelman. In this case the theorem fits the situation beautifully. However one will probably have to make some assumptions to recognize forms such as

$$f(x^c y) \quad (bxy'-a) = x^a y^b \quad (xy' + cy)$$

In order to perform the integration, y in G(x,y) is replaced by $\frac{u}{x^n}$. It is then hoped that SCHVUOS can rid the resulting

expression of all occurrences of x.

Examples (see appendix E for further discussion of these examples)

1) $(x-x^2y)\ y'-y = 0$

becomes

$$\frac{du}{u\left(1+\frac{1}{1-u}\right)} - \frac{1}{x}\ dx = 0$$

2) $xy' + y \log_e x - y \log_e y - y = 0$

becomes

$$\frac{du}{u\left(-1 - (\log_e u - 1)\right)} = \frac{dx}{x}$$

In Appendix E we describe an experiment in which SOLDIER was asked to solve 76 differential equations selected from a college text. SOLDIER was able to completely solve 67 of these problems with an average time on the order of 5 records. An analysis of the problems it failed to solve and steps taken to improve SOLDIER's performance on some of these problems is also given in Appendix E.

We would also like to mention the existence of a program which solves linear differential equations of any order with constant coefficients (see Engelman [36]). It was written by Ernst for the MATHLAB system. It utilizes the Laplace Transform method for solving such equations. The program makes use of the rational function package of the MATHLAB System.

Some methods which were not described above should be pointed out. There are many special cases of integrating factors which can be considered. In particular, one method guesses the form of the integrating factor to be $x^a y^b$, substitutes that form

into the equation and solves the linear equations in the parameters that result after setting up the conditions for exactness (i.e., $\frac{\partial}{\partial y}(\mu M) = \frac{\partial}{\partial x}(\mu N)$). If the system of equations can be satisfied, then Method 3 (Exact) is applied. If the differential equation contains a subexpression which is irrational in both a and y (e.g., $\sin(x^2 + y^2)$), then it might be useful to substitute for some part of this subexpression (e.g., $u = x^2+y^2$). One can also attempt to switch the independent and dependent variables. Such a change would be useful in

$$(xy + x^2)\, y' + e^y = 0$$

since it leads to the Bernoulli differential equation

$$e^y\, x' + xy + x^2 = 0$$

There is a large body of knowledge regarding Ricatti and Abelian equations (i.e., $y' = f(x)y^2 + g(x)y + h(x)$, and $y' = f(x)y^3 + g(x)y^2 + h(x)y + k(y)$). These methods, however, frequently rely on knowing one or more particular solutions to the differential equation. Information regarding methods applicable to Ricatti and Abelian equations and to more general differential equations can be found in Kamke. Kamke also contains a table of about 1250 equations whose solution is frequently given in some detail.

As is pointed out in Appendix A, a great deal of the information about differential equations could be stored in tables and searched by computers. If we presume that a continual effort will be made to generate a library of programs and tables for differential equations, then programs will become a formidable tools for solving these problems.

CHAPTER 7

CONCLUSIONS AND SUGGESTIONS FOR FURTHER WORK

The Performance of SIN

We believe that SIN is capable of solving integration problems as difficult as ones found in the largest tables. The principal weakness of SIN in relation to these tables is in cases of integrands which contain variable exponents and which usually result in solutions which are iterated integrals. Edge can solve some of these integrals (e.g., $\int x^n \cos x \, dx$) since it contains special checks for variable exponents. However none of SIN's methods in stage 2 are able to obtain such iterated integrals. The experiment reported in Appendix D also showed SIN's weakness in handling certain algebraic integrands. On the other hand the power of MATHLAB's rational function package means that SIN is able to integrate many problems not present in the tables. Decision procedures for cases such as the Chebyschev integrals give SIN a capability which is not present in most tables.

SIN appears to us to be faster and more powerful than SAINT. The added power of SIN is principally due to the additional methods that SIN possesses. The additional speed is gained by the change in the organization of SAINT and by the use of tighter progress requirements. In Appendix C we pointed out that though SIN can solve problems solved by SAINT two orders and frequently three orders of magnitude faster than SAINT, that this figure is deceptive. It is probable that under optimal conditions for SAINT and SIN these figures will reduce dramatically so that the gain in speed will average to about a factor of three. In

cases where the Derivative-divides routine is successful in solving a problem (about half the time), the ratio should be much higher. The average will be lowered by the increased effort spent on algebraic manipulation on the other problems. SIN's simplifier SCHVUOS, is probably a good deal slower (but more powerful) than SAINT's hand-coded simplifier. This factor affects the cost of most of the other processes such as differentiation and matching.

On the Organization of SIN

Instead of describing the organization of SIN at this point, we would like to indicate certain aspects of this organization which arise out of the discussion in Chapter 4. The reader is referred back to Chapter 2 for an outline of SIN's organization.

One of the difficulties that AI programs will increasingly face involves communication (see Newell [46]). If a subroutine performs an analysis of a problem then its analysis must be communicated to its parent routine in such a manner that the parent routine can easily understand the information. If two subroutines are working in parallel, one may need to know what the other one is doing in order to perform efficiently. An example of the usefulness of the latter type of communication was pointed out in Chapter 4 in the section in which we described SAINT's solution of $\int \frac{x^4}{(1 - x^2)^{5/2}}\, dx$. Here it was noted that in one of the subproblems SAINT should not have performed the substitution $y = \tan\frac{1}{2}x$ since another trigonometric substitution on the problem had already been made which was undoubtedly superior. In this case SAINT did not seek out the necessary information. A similar difficulty arose

when SAINT's methods could have performed transformations which were the inverse of previous transformations. This occurs in the method which substitutes $\frac{\sin x}{\cos x}$ for tanx, since this method may later substitute $\frac{\tan x}{\sec x}$ for sin x. In this case SAINT did communicate the existence of the previous transformation. While we do not wish to minimize the need for explicit communication in complex problem solving programs, we do want to point out the usefulness of highly implicit communication in certain situations. If a parent routine knows enough about the operation of its subroutines, then it is not necessary to communicate a great deal of information, the parent routine can determine what has probably occurred with just a few key works of exchange. We think that such implicit communication occurs when FORM finds excuses for the failure of its methods to solve certain problems. In fact in these cases the methods are not aware of the situation as much as FORM is. SIN will not attempt the $\tan\frac{1}{2}x$ transformation if another trigonometric transformation is possible since this choice was built into the program. Similar remarks hold for the trigonometric identity transformation. What these examples appear to point out is that when one is able to centralize control in a routine which has sufficient understanding of a task, then the communication requirements in the program are markedly reduced.

We noted in the discussion in Chapters 2 and 4 that SIN employs tighter progress constraints than does SAINT. This implies that there may be some problems which SIN will not attempt to handle though it has sufficient machinery for solving them. (On the other hand, we believe that SAINT will attempt to solve $\int\frac{\sin x}{x}\,dx$ until it runs out of time or space.) We are not particularly worried by such occurrences. It appears

to us that it is more important at present that a program have a good understanding of what it is able to do rather than that it have a mediocre understanding and be able to solve more problems. If one desired to increase the power of SIN we would wish that he spend the effort on improving the analysis done by FORM rather than that he spend it on increasing the search in FORM. We understand, of course, that it is not always possible to take this approach. The domain of nonlinear differential equations is a good example of such a situation.

On the Organization of SOLDIER

We noted in the Introduction that we did not expect to find a concept as powerful as the Edge heuristic in the domain of first-order, first-degree ordinary differential equations. Thus we were not surprised to fail to find a practical method similar to Edge. In fact the most notable aspect of SIN's organization that we carried over was the reliance on tight progress constraints. It seems to us that human analysis of this problem domain also employs tight progress constraints in the solution methods.

Let us recall from Chapter 6 that SOLDIER employs eight solution methods. These methods are attempted one at a time. If a method decides that it is able to make a simplifying transformation (i.e., a direct reduction to integration or a reduction to a known and simpler differential equation form), then it will attempt it, and the result of the transformation will be the value of SOLDIER. Otherwise the next method will be considered.

In Appendix E we tested SOLDIER on some problems given in a differential equations text. SOLDIER was able to solve 67 out of 76 of these problems. We do not believe that one should conclude from this perfor-

mance that SOLDIER is far removed from being as powerful a differential equation solver as expert humans are. We think that if the improvements and extensions to SOLDIER that we suggest in Chapter 6 and below are made then SOLDIER will be a powerful program indeed. We were disappointed when we recognized this to be the case. The reason for it is that mathematicians have not made great advances in this problem domain over the past three hundred years.

On the Applications of LISP

Unfortunately, and mainly wrongly, LISP has acquired the reputation of being a language with very low execution speed. One factor leading to this reputation is the slow speed of arithmetic in most LISP implementations. (The Hawkinson-Yates system for the 7090 is an exception.) Yet when one declares variables to be fixed or floating it is possible for LISP to execute arithmetic statements as well as any other processor. It is the convenience of mixed data types (during execution) which forces the slow, interpretive execution speed of arithmetic operations in LISP. Another factor leading to this reputation is that old and famous programs such as SAINT ran interpretively. Compilation usually results in approximately a twenty fold gain in speed. However the largest factor leading to this reputation is due to the attitude of the LISP programmers. LISP programs were usually developed in research projects where speed was only a minor consideration. (It is safe to say that many impressive programs such as Bobrow's STUDENT [4], Evans' ANALOGY and Slagle's SAINT could not have been written as doctoral dissertations except in LISP.) The trend in the recent past has been toward using LISP as a practical language

for projects with real time constraints on response. For example the MATHLAB system of Engelman and the robot projects at MIT and STANFORD have such real time constraints. It is thus important to recognize that LISP programs can be written which are relatively fast provided that one takes speed into consideration in designing the programs. It is our hope that SIN can serve as a model for this lesson and remove some of the stigma attached to LISP. It is far too easy to write LISP programs which execute slowly if one becomes beguiled by the ease of using LISP's recursive mechanisms. SAINT's pattern matching program Elinst was far too recursive to run efficiently. However it was a much smaller program thereby and this factor was crucial in the implementation of SAINT. The rational function package used in SIN runs slowly when parameters are introduced into a rational function. While such a decrease in speed is inherent in the task, it is also due to the extensive utilization of the recursive nature of the LISP list structure in the representation of rational functions. A special purpose representation of rational functions such as used in Brown's ALPAK [6] or Collins' PM system [12] should increase the speed of the rational function package by one to two orders of magnitude.

On the Teaching of Integral Calculus

We would like to see the introduction into first year calculus courses of the concepts underlying the Edge heuristic and the Liouville Theory. Besides giving the student a very powerful integration method, such a study might acquaint him with practical applications of notions derived from modern logic such as Godel numbering or decidable problem domains. Such a course might also indicate why $\int e^{x^2} dx$ is not an ele-

mentary function rather than leave such a statement without proof. The relationship of the Edge heuristic and the problem solving technique of guessing could reasonably be emphasized in courses aimed at a more practical foundation.

Improvements and Extensions to SIN and SOLDIER

All the programs discussed in this thesis would profit by being rewritten for the LISP system of the MAC PDP-6. The PDP-6 LISP system executes about three times as fast as the 7094 LISP system on compiled function and even faster on interpreted ones. This is due to the improved instruction set of the PDP-6 and to improved system's programming rather than an increase in the machine speed. The MAC PDP-6 also has 256 K of memory which would mean that all the routines could certainly be loaded at one time. This would allow greater interchange between SIN and SOLDIER and the rational function package. It would allow SIN and SOLDIER to be used as subroutines to the MATHLAB system of Engelman. The excellent scope output routines of Martin [37] are available on the PDP-6 as are teletype output routines written by Millen for the MATHLAB System [40]. Routines which accept FORTRAN-like (i.e., infix) notation for algebraic expressions are available and should be used instead of the LISP (i.e., prefix) notation which is now used in inputs to SIN and SOLDIER. Anderson of Harvard University is currently working on a program which permits hand written input of algebraic expressions from a Rand Tablet [1]. Such a program could be used in the future as well.

SCHATCHEN should be rewritten so that new modes can be defined by the user without reprogramming relevant sections of SCHATCHEN. The simplifier SCHVUOS served us well while we required a small simplifier. However a new, more powerful and efficient simplifier written along the

lines indicated in Chapter 3 should be used. As is clear from Chapter 6 and Appendix E this simplifier should have factoring and division capabilities not currently available in general purpose simplifiers. The task of matching expressions for identity should be performed by a program such as Martin's matching program rather than by SCHATCHEN [37].

SIN's second stage would profit from a better handling of algebraic integrands. This is clear from Appendix D. Another lesson learned in that appendix is the usefulness of a capability whereby the user can communicate with FORM and some of the methods used in SIN in order to introduce new functions such as the error function. A table of integrals involving the error function which contains 145 entries was computed by Maurer in 1958 [38]. Such a table should be computable by SIN as well.

It is clear that much more work needs to be done on the Edge heuristic both as a method for solving integration problems and as a possible tool for teaching freshman calculus students. We understand that Risch is currently programming his method of integration using the rational function package. Such a program could be included in SIN's third stage as well.

In discussing SOLDIER in Chapter 6 we noted that a great number of methods are known which have not yet been programmed. An interesting project is involved in finding particular solutions to differential equations. Such solutions can be used to find general solutions to Ricatti differential equations. In Appendix E we noted that the output of SOLDIER rarely conforms with the form of the text books' output. Another project would be to devise a routine which translates SOLDIER's output to conform with the implicit conventions used in text books.

We believe that if work is continued on the implementation of new methods for SOLDIER, then this program will become a truly formidable tool in solving ordinary differential equations. In fact a program such as SOLDIER can become an active competitor with text books or journal articles as a medium for the permanent storage of knowledge about methods of solution.

On a Mathematical Laboratory

In a forthcoming monograph by Martin and Moses the concept of a mathematical laboratory will be introduced. In a mathematical laboratory a user will be able to solve symbolic problems in mathematics. A mathematical laboratory is envisioned to consist of two major components, a general purpose system and a set of specialized programs. The general purpose system will deal with input and output and will provide a command-oriented language with many capabilities. The specialized programs will deal with tasks which are sufficiently complex to require a separate organization. SIN and SOLDIER are prototypes of such specialized programs. Specialized programs will in the future employ a set of rather general routines such as a pattern directed language similar to SCHATCHEN or a simplifier such as SCHVUOS. These frequently used routines will form a data base from which new specialized programs will be more easily written in the future. Work is proceeding in this country on all aspects of such a mathematical laboratory, but we shall concentrate our discussion on the specialized programs. In a recent thesis [28], Iturriaga has written a program in FORMULA ALGOL for finding limits of expressions and for determining whether one expression is greater in value than another over some domain. This work represents an extension of work on

limits performed by Fenichel [19]. No work has been done to our knowledge, on finding sums of infinite series. Jolley provides a table of such series [29]. Nor has any significant work been done on definite integration. Bierens de Haan's monumental work on this area can be consulted [24]. In both of these cases one might at first utilize a table look up as described in Appendix A.

Leaving aside the area of analysis we note that Maurer [39] and McIntosh [57] reported on systems which deal with finite groups. Some routines have also been written for solving specialized tasks in topology. In fact a new theorem in topology was proved as a result of experiments performed by such programs [50]. Likewise specialized programs in combinatorics have been written [16]. Such programs should be expanded upon, systematized, and made available as part of a larger symbolic manipulation system in pure mathematics.

Along with the need for practical work in algebraic manipulation there is a need for parallel work on theoretical results. Collins' study of the Greatest Common Divisor algorithm led to a major imporvement of the Euclidean GCD method [13]. Similar studies are needed of methods for factoring polynomials, especially over extensions of the ring of integers. We need a study of the degree of growth of the results of certain algebraic transformations. We should have examples of very bad problems. In [42] we present such a problem in the domain of polynomial equations. Recursively unsolvable results such as those in Appendix B point out certain difficulties in algebraic manipulation. Proofs of the decidability of certain subcases such as in Richardson [52], Caviness [9], Brown [7], Risch [53], and Tobey [63] are useful also and these may in turn lead to

programs which implement the decision procedures used.

On Artificial Intelligence

In the area of Artificial Intelligence we would applaud all projects which required and utilized a large base of specialized knowledge. Robot projects are examples of such projects. On a less ambitious level we would like to note that it might be useful to develop a program which solves word problems in the calculus. Such a program would counter, (if only temporarily!) the objections of those who claim that the semantic approach of Bobrow cannot be extended. One approach toward this problem would be to construct several methods of solution (e.g., "rate" problems of several types). Then the program would use local clues (probably key word analysis as in Weizenbaum's Eliza [66] will do) to determine which solution method is appropriate. Then the method chosen should guide the program in extracting the information from the problem statement necessary for a complete solution.

It would also be interesting to have some work leading toward a program which solves multiple choice questions on the level of the MAA high school prize examinations. Let us consider a typical problem.

"At what time between 4 and 5 PM are the hands of the clock exactly opposite each other?"

If the program knows that the answer involves the denominator of 11 and one such answer is presented, then it should guess that answer. If only one answer involves a denominator of 11 and is moreover between 4:50 and 4:55 PM, the program should guess it. These guesses would be made at stage 1 of the program.

If stage 1 is not effective but if the program knows the method of solution (a linear equation), then it should solve the equation. This would be done at stage 2 of the program.

If neither of these stages is appropriate, then the program must obtain an analysis of this situation. Such an analysis is presently beyond the capabilities of AI programs, but not grossly beyond these capabilities.

Presumably one of the methods available to this program is a rate problem solver. The statement of the problem does not immediately imply a rate problem but the knowledge that the minute hand and the hour hand travel at different rates could lend weight to such an hypothesis. Let x be the time in minutes past 4 o'clock at which the event occurs. Then the minute hand travelled x minutes between 4 o'clock and the occurrence of the event. The hour hand travelled $\frac{x}{12}$ minutes during that time. However the hour hand started with a 20 minute advantage and ended thirty minutes (one half a revolution) behind. Thus

$$x = 20 + 30 + \frac{x}{12}$$

$$x = \frac{600}{11} = 54\frac{6}{11} \text{ minutes}$$

The solution above required the use of information about clocks and the relationship between clocks and circles. It also required a sophisticated word problem solver that was able to utilize this information to set up the linear equation. Another method of solving this problem relies somewhat more heavily on making inferences about diagrams. In either case it appears that a good deal of machinery is required for the analysis of this problem. Besides the word problem solver a program which makes inferences based on diagrams of plane figures is also useful. While such programs may not be sufficient in order to perform the analy-

sis of this problem, they certainly go a long way in that direction.

APPENDIX A

ITALU - AN INTEGRAL TABLE LOOK - UP

This appendix describes some experiments which were performed with an integral table look-up. Although a table look-up is probably inferior in the long run to an integration program with regard to power or speed, the techniques employed in this routine could be found useful in other areas of symbolic mathematics such as exact definite integration, summation of series, or differential equations.

There are several ways in which one could search a table of integrals. There is the brute force approach. In this case each entry in the table is matched for equivalence with the expression to be integrated. This scheme is used in SIN's Derivative-divides routine. Such a scheme takes a long time when the table is large, of course. A better approach is to sort the entries in the table by the factors which appear in them (e.g., all entries with sin x as a factor are in one subtable). Thus when presented with $\sin x e^x$, one checks all subtables for the one which contains sinx. In that subtable one checks for another part of the table which contains $\sin x e^x$ and there one presumably finds the entry desired. This approach would require that there be n! entries for an integrand with n factors (unless the expressions are canonically ordered). A table look-up along these lines was discussed in Klerer and May [32].

Besides being relatively slow these approaches are not sensitive to the fact that an integral table usually presents generalized forms of integrands (e.g., $\sqrt{ax^2+bx+c}$) and not just particular integrands. (e.g., $\sqrt{x^2+1}$). This is due to the presence of undetermined constants in the integrand. These constants are used as coefficients as in $\int \sin(ax+b)dx$ or exponents as in $\int x^n dx$ or $\int x^n \sin x dx$. The example $\int x^n \sin x dx$ points out a further feature of the integral table, that is, the presence of iterated integrals in the table. A good integral table look-up should be required to make use of all of these features of the tables.

An integral table look-up, called ITALU, was programmed to account for the features of the table just mentioned. It had the additional property of being relatively fast by making use of the technique of hash-coding.

By carefully hash-coding the expression to be integrated one can expect to obtain a number which would correspond to relatively few expressions in the table. Furthermore the hash-code can be designed to account for the distinctive features of the table. The hash-coding scheme which was implemented ignored constants in sums and products. Thus sin (ax+b) coded the same as sin(2x), sin(x+2), sinx, and $\sin(3\pi x+5y+z)$. The hash-code, moreover, was a floating-point number and the code of a sum was the sum of the codes of the terms in the sum, with a similar rule for products. Thus the code maintained the algebraic identities for sums and products. Hence $\sin x e^x$ coded like $e^x \sin x$. In this manner we avoid the need for

a canonical form of an expression. One further feature of this coding scheme was that terms in a sum which had codes identical with those of previous terms were ignored. Thus sin (x+yx) coded like sinx and $x^2+2xy+3x$ coded like $(2y+3)\ x + x^2$ and ax^2+bx+c.

The coding scheme was obtained recursively. The variable of integration had a fixed code of 0.95532. Any trigonometric, arctrigonometric or logarithmic function had associated with it a fixed floating-point constant which generally was exponentiated by the code of its argument in order to obtain the code of the expression. Sums and products were treated as described above. Exponentiation was a relatively complex operator for the coding scheme. This is due to the frequent occurrence of exponents -2, -1, $-\frac{1}{2}$, $\frac{1}{2}$, 2 in the tables. When these exponents occurred the code for the base was raised to the exponent and the result was the code of the expression. Any other constant exponent was coded as 1.43762 and the value of the subsequent exponentiation became the code. Thus x^n is coded like x^3 or x^a or $x^{-4.5}$. Fixed bases were all coded alike. Thus e^x coded like 2^x or y^x.

An advantage of this coding scheme was that SCHATCHEN patterns could be coded easily as if they were expressions. This was due to the fact that the variables in the pattern were considered constants with respect to the variable of integration (assumed to be x throughout the table), and hence were ignored in sums and products and had a fixed value in exponents. Entries in the tables had

integrands which were SCHATCHEN patterns (e.g., sin (^{A}/COEFFPT, NONZERO-AND-FREEOFX $^{x+B}$/COEFFP, FREEOFX). Thus the full matching capability of SCHATCHEN could be employed in order to obtain the values of the constants in the integral table entry.

ITALU had an internal table of code numbers for the expressions in the table. This internal table was searched using a binary search (i.e., the codes were linearly ordered by their numerical values). Corresponding to each code in this table was the location on the disk where the integral table entry resided. Once a code was assigned to an expression, it was determined if an entry in the table had an identical code, and the file on the disk containing that entry (if any) was read. In order to conserve disk space several entries were on the same file, but these entries were associated with their codes so that the search of the file was linear but rapid. For each expression having the desired code (several are possible), SCHATCHEN was used to determine if there was a match between the pattern which represented the integrand in the table and the original expression. If no match was found, the next expression was examined, and so on until all the expressions with the appropriate code were examined. If a match was obtained, the integral was evaluated after making appropriate substitution for the result of the match. Thus the integral contained the values of the constants in the integrand. The device of evaluating the the integral allowed the integral to be a LISP function. In this

manner iterated integrals could be obtained. Hence the ITALU program satisfied the requirements of an integral table look-up that we considered above.

The implementation of ITALU was carried through up to the point where all of the steps above had been implemented and the program was tested on several problems. The largest number of entries in the table was only ten at any given time, and thus the properties of the coding could not be fully assessed (e.g., one could not tell how frequently unrelated entries yielded the same code number). The execution time of a call to ITALU was generally about 1 second. Most of this time was spent accessing and reading the disk. A set of routines were written for facilitating the addition of new entries to the table. However the description of each entry as a SCHATCHEN pattern with a corresponding integral was a fairly tedious job. A compact representation of the expressions in the table was obviously desirable, but was not implemented.

Modifications to the hash code of ITALU were considered. Under the current coding scheme $\sqrt{x^2+1}$ codes like x. One possibility is to ignore the value of constants in sums and products, but recognize their existence. Such a scheme would be useful in handling algebraic expressions.

We also considered using a hash-coding scheme, such as Martin's [37]. Martin's hash codes are elements of finite fields rather than floating point numbers. Finite field

arithmetic is preferable when there is a risk of a floating-point overflow or a round-off error during the computation of the hash code. We felt that these difficulties could be ignored or easily overcome in the coding of expressions to be integrated. In order to account for round-off errors, we thus allowed for a variance of 1×10^{-6} between the code of an expression and one in the table.

In the domain of symbolic integration, a table look-up is probably not the best solution. Programs can now compete effectively in many cases with the tables with regard to speed and completeness. The situation in the future can only improve the relative position of the integration programs. Tables such as Petit Bois' [51] with its 2500 entries contain many errors, some of which are serious (e.g., $\int \log \cos x dx = \frac{1}{\cos x}$, [51] p. 150).

However table look-up devices appear to have current usefulness in other areas of symbolic mathematics. Very little work is being done at present on summation of series and exact definite integration. Tables in these areas exist - Jolley's [29] in summation and Bierens de Haan's [24] monumental work on definite integration. For differential equations we reported solutions methods in Chapter 6. However much still remains to be done, and tables could be used as long as programs have not caught up with the full power of tables such as Kamke's [30]. Tables could be extended to include a great deal of information besides exact solutions. For example, tables could be employed to obtain good numerical techniques for solution or references to papers on

particular cases. We should point out that some entries in a table would be hard to look-up in any reasonable way. For example, the entry $xy'=yH(x^{n}y)$ properly deserves a special purpose program as was done in Chapter 6. Information about chemical compounds is currently being stored in tables which are searched by specialized techniques. Similar methods could be used in mathematics. The exact methods of ITALU are clearly not extendable to the other problem domains - special purpose programs should be used in each case. However the hash-coding technique coupled with the use of a matching program for increased power seem relevant to each of the areas considered.

APPENDIX B

RECURSIVELY UNSOLVABLE RESULTS IN INTEGRATION

A recent theorem by Richardson [52] showed that the matching problem for a class of functions we shall call R-elementary is recursively unsolvable. This result is easily applied to show that the question of determining whether integrals of R-elementary functions possess R-elementary solutions (or elementary solutions in the sense of Liouville (Chapter 5)) is likewise recursively unsolvable. Richardson's result, announced January 1966, is probably the first theorem about recursively unsolvable problems in analysis and has aroused great interest in the field of algebraic manipulation. References to it are made in Brown [7], Caviness [9], Fenichel [19], Moses [42], and Tobey [63].

There is, however, a feeling among some (e.g., Risch [53]) that Richardson's unsolvability result may be due to the fact that the integration problem he showed unsolvable is not well-posed. In this appendix we shall sketch Richardson's unsolvability proof and indicate points in the proof where some of this contention has arisen. We shall then present results of a similar nature to Richardson's which avoid these difficulties in the proof by extending the domain of the problem to nonlinear differential equations. These results are proved using similar techniques to Richardson's and were originally proved, interestingly enough, over a year before Richardson announced his proof.

In order to proceed we shall require the following definitions.

The R-elementary functions are obtained by the operations of addition, multiplication, division and substitution upon real variables, $x_1, x_2, \ldots, x_n$ using the constants π, the rational numbers, $\log_e 2$, and the functions e^x, sin x, cos x, and $\log|x|$.

The constant problem is to decide, given an R-elementary function f(x), whether f(0)=0.

The identity(matching)problem is to decide, given an R-elementary function f(x), whether $f(x) \equiv 0$.

The integration problem is to decide, given an R-elementary function f(x), whether there exists an R-elementary function g(x), such that $g'(x) \equiv f(x)$.

Richardson first showed that the identity problem reduced to solving the constant problem. Thus, if one restricts the R-elementary function to a domain where the constant problem is presumably solvable (e.g., by allowing only the rational operations), then the matching problem is likewise solvable.

He then showed that the matching and integration problems for the R-elementary functions is recursively unsolvable. In order to proceed with our sketch of that proof, we shall require the following definitions.

Hilbert's 10th Problem (The Diophantine Problem)

Does there exist a procedure for determining whether the

equation $P(x_1, x_2, \ldots, x_n)=0$, where P is any polynomial with integer coefficients, has a solution where each x_i is an integer?

Exponential Diophantine Problem

Does there exist a procedure for determining whether the equation $P(x_1, x_2, \ldots, x_n, x_{n+1})=0$, where P is any polynomial with integer coefficients and where x_{n+1} is replaced by 2^{x_1}, (i.e., $P(x_1, \ldots, x_n, 2^{x_1})=0$) has a solution with each x_i, i=1, ..., n an integer?

Theorem (Davis, Putnam, Robinson) [14]

The exponential diophantine problem is recursively unsolvable.

The version of the Davis-Putnam-Robinson result that Richardson used is as follows:

Theorem A There exists a polynomial $Q(y, x_1, \ldots, x_n, 2^{x_1})$ such that the problem of determining whether for each integer value of y there exist integer solutions $x_1, \ldots, x_n$ to the equation $Q(y, x_1, \ldots, x_n, 2^{x_1})=0$, is recursively unsolvable.

Hilbert's 10th problem has not yet been decided although it is suspected that the problem is recursively unsolvable as well.

Let us now proceed with Richardson's argument.

Consider the polynomial Q of Theorem 1. Let the x_i be real numbers. Then, if the equation I

$$\text{(I)} \qquad \sum_{i=1}^{n} \sin^2 \pi x_i + Q^2(y, x_1, \ldots, x_n, 2^{x_1}) = 0$$

possesses real-valued solutions for an integer value of y, then the x_i must be integers, and if Q possesses integer solutions, equation I certainly has real solutions.

Note that since each term in I is real-valued, the "sum of the squares" device forces each term to be zero. Since $\sin\pi x_i = 0 \leftrightarrow x_i$ is an integer, the x_i must all be integers. This illustrates a concept we shall call <u>forcing</u>. Forcing will be frequently used in this appendix. The term $\sum_{i=1}^{n} \sin^2 \pi x_i$ forces Q to possess integer solutions. The use of π and sin x in this manner was foreshadowed by Tarski [61].

The next step is to show that there exists and R-elementary function $f(y, x_1, \ldots, x_n)$ such that $f(y, x_1, \ldots, x_n) < 1$ for a given integer y and for some real x_i if and only if $Q(y, x_1^*, x_2^*, \ldots, 2^{x_1^*}) = 0$ for some integer values of the x_i^*, and for the same integer value of y.

Richardson shows that we can take $f(y, x_1, \ldots, x_n)$ to be of the form

$$A(n)\left[\sum_{i=1}^{n} \sin^2 \pi x_i \; K_i^4(y, x_1, \ldots, x_n) + Q^2(y, x_1, \ldots, x_n, 2^{x_1})\right]$$

where A is a large R-elementary function of n and each K_i is a suitably chosen large R-elementary function of its arguments. In this form f is an R-elementary function. The proof that f has the desired

property utilizes an argument based on the consideration that if f is sufficiently close to 0 in value, let us suppose that $f(y, \bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n) \leq 1$, and let each $\bar{x}_i$ be close to the integer, x_i^*, say, then $Q(y, \bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n, 2^{\bar{x}_1}) < \frac{1}{A(n)}$. What is desired is to force $\mathcal{Q}$ to have the value 0 at the x_i^*. Since $\mathcal{Q}$ is continuous in its variables (it is a polynomial in them) and moreover has integer values for integer arguments (the coefficients are integers), what is necessary is that the derivative of $\mathcal{Q}$ is sufficiently small so that Q does not materially change its value on the interval between $\bar{x}_i$ and x_i^*. For this purpose the K_i which are based on the partial derivatives of Q are forced to be small as well. This is done by requiring $\sin \pi x_i K_i^2 \leq \frac{1}{A(n)}$.

Now Richardson shows that one can obtain a coding which reduces the problem for the n variables x_i of $\mathcal{Q}$ to a single variable x. He obtains a function G(y, x) such that $G(y, x) < 1$ for real

$x \leftrightarrow (\forall \epsilon > 0)(G(y, x) < \epsilon) \leftrightarrow \exists$ real x_i

$f(y, x_1, \ldots, x_n) \leq 1 \leftrightarrow \mathcal{Q}(y, x_1^*, \ldots, x_n^*, 2^{x_1^*}) = 0$ for some integers x_i^*.

The coding is

$$x_1 = h(x),\ x_2 = h(g(x)),\ x_3 = h(g(g(x))),\ \ldots$$

where $h(x) = x\sin x$, $g(x) = x\sin x^3$.

Richardson now uses the $\log|x|$ function to obtain a decision. Consider the following equations:

$|x| = e^{\log|x|}$, thus the absolute value function is R-elementary.

$x \dot{-} y = \frac{x-y+|x-y|}{2}$, this subtraction has value 0 if $y=x$.

$\text{Min}(y,x) = y \dot{-} (y \dot{-} x)$, the minimum function restricted to non-negative values.

Now if $G(y,x) \leq 1$ for some real x and integer y, then $G(y,x) < \frac{1}{2}$ for some real x by the ϵ case above, and for this x, $2 \dot{-} 2G(y,x) > 1$. Thus, $\min(1, 2 \dot{-} 2G(y,x)) = 1$ for some real x. If $G(y,x) > 1$ for all real x, then for all real x, $\min(1, 2 \dot{-} 2G(y,x)) = 0$. By the continuity of G which is preserved either $\min(1, 2 \dot{-} 2G(y,x)) \equiv 1$ for some interval of values on the real axis for x and for a fixed integer value of y, or $\min(1, 2 \dot{-} 2G(y,x)) \equiv 0$ for all real x.

Now if we let $M(y,x) \equiv \min(1, 2 \dot{-} 2G(y,x))$, then the question of deciding whether $M(y,x)$ is identically 0 is equivalent to deciding whether $Q(y, x_1, \ldots, x_n, 2^{x_1}) = 0$ has integer solutions and is thus recursively unsolvable. $M(y,x)$, we note, is R-elementary.

The above is a sketch of the proof of the recursive unsolvability of the matching problem. The recursive unsolvability of the integration problem is obtained as follows:

Consider

$$\int M(y,x)e^{x^2}dx$$

If $M \equiv 0$ for some integer value of y, then the integrand is 0 and possesses a solution (e.g., 0). If $M \equiv 1$, on some interval then

the integrand is equivalent to e^{x^2} which possesses no elementary solution on any interval, as is well-known. Hence, the integration problem for R-elementary functions is unsolvable since one cannot tell whether $M \equiv 0$.

This completes the sketch of Richardson's proof. As was seen, the decision step in the matching problem necessitated the use of the absolute value function. Caviness argues that either the absolute value function or the constant π (used in $\sin \pi x$ and needed to assure a zero value on integer arguments) are the culprits in allowing Richardson's results to hold. The constant π should not be too surprising in the context since there are many problems related to the constants e and π which are not yet solved (note $\sin \pi x = \frac{e^{i\pi x} - e^{-i\pi x}}{2i}$). For example, it is not known whether $e+\pi$ is a rational number.

We should note that the absolute value function arose when we considered only one of the infinite number of inverses to the log function. For example we can obtain the absolute value function by considering $\sqrt{x^2}$ to possess only one solution. If we were to evaluate each of the values of an R-elementary function and were to consider f(x) to be equivalent to 0 if it were 0 for each of its values, then one might obtain a more tractable problem. One would still be left with ticklish problems regarding the constants e and π. These one might suppose are not very interesting from a practical

standpoint. However, by introducing square-roots into the picture, one might complicate the situation once more since such an introduction appears to lead to difficulty in integration (see Ritt [54], Risch [53]).

The recursive unsolvability of the integration problem was obtained by making use of the fact that one could not tell what the simplest description of the integrand was. In previous work on the problem of integration in finite terms such a difficulty was usually ignored. If one could ignore such a difficulty in the matching or in the constant problem, then these problems would disappear. The same cannot be said of the integration problem, of course.

The question now arises as to whether there are unsolvable problems in the area of symbolic integration which avoid the use of the absolute value function and which do not simply reduce to the matching problem. Below we give some simple and hardly surprising results which indicate that such problems do exist when one considers nonlinear differential equations.

We shall require the following result:

Theorem B (see Ritt p. 73)

The equation

$$(II) \qquad y' + y^2 = 1 + \frac{p(p+1)}{x^2}$$

where p is a constant (a computable complex number, say), has a particular solution which is a rational function in x (with computable

complex coefficients) if and only if p is an integer.

Theorem 1

The exponential diophantine problem (Theorem A) is equivalent to the problem of determining whether, for integer values of y, the system of differential equation **S** has particular solutions which are rational function in x.

(Hence, the latter problem is recursively unsolvable.)

$$(S)\quad \begin{aligned} &a)\ \frac{dp_i}{dx} = 0,\ i=1,\ \ldots,\ n \\ &b)\ \frac{dy_i}{dx} + y^2 = 1 + \frac{p_i(p_i+1)}{x^2},\ i=1,\ \ldots,\ n \\ &c)\ \frac{dz}{dx} + z^2 = 1 - \frac{Q^2(y,\ p_1,\ \ldots,\ p_n,\ 2^{p_1})}{x^2} \end{aligned}$$

Proof. Suppose S has such a set of solutions for a given integer value of y.

By a) each p_i is a constant.

By b) and Theorem B each p_i is an integer.

$Q(y,\ p_1,\ \ldots,\ p_n,\ 2^{p_1}) = 0$ by c) for y an integer.

This is so since by a) and b) Q is a constant. Thus, for z to have a particular solution which is a rational function, $-Q^2 = q(q+1)$ for some integer q. But $q(q+1) \geq 0$ for integers q and $-Q^2 \leq 0$ since Q is integer valued. Thus, $Q(y,\ p_1,\ \ldots,\ p_n,\ 2^{p_1}) = 0$ for integer

values of $p_1, \ldots, p_n$.

Suppose Q did possess integer solutions c_i for some integer value of y, then by fixing each p_i to be the corresponding c_i, we obtain a set of rational solutions for S.

Theorem B has a corollary which states that the differential equation II has a general solution which is a Liouville function if p is an integer.

Theorem 1 can, therefore, be extended to show that the problem of determining whether systems of differential equations of the form S have solutions which are Liouville functions is recursively unsolvable.

Let us consider the diophantine analogue of the system S (i.e., no exponentiation in Q). We now have a system of polynomial equations with integer coefficients. The solutions of such systems of equations is in the domain of differential algebra (see Kaplansky [31]). Theorem 1 leads to the result that Hilbert's 10th Problem reduces to a decision problem in differential algebra.

Let us now consider the problem of determining whether a differential equation $f(x, z, z', \ldots, z^{(n)}) = 0$ has a solution $z(x)$ where z and all its indicated derivatives are real-valued functions of x.

More precisely consider

$$g(y, x, z', \ldots, z^{(n)})$$

$$= z^{(n)2} + Q^2(y, w_1, w_2, \ldots, w_n, 2^{w_1}) + \sum_{i=1}^{n} \sin^2 \pi w_i$$

$$= 0$$

In g, y is an integer, x is the independent variable and is real, z is the dependent variable and the w_i are defined as follows:

$$w_n = \frac{z^{(n-1)}}{(n-1)!}$$

$$w_{n-1} = \frac{z^{(n-2)} - xz^{(n-1)}}{(n-2)!}$$

$$\vdots$$

$$w_1 = z - xz' + \frac{x^2 z''}{2!} + \ldots + (-1)^{(n-1)} \frac{x^{(n-1)} z^{(n-1)}}{(n-1)!}$$

Theorem 2 The problem of deciding whether $g(y, x, z, z', \ldots, z^{(n)}) = o$ has a real-valued solution which possesses n real-valued derivatives is recursively unsolvable as y varies over the integers.

Proof. Let y be fixed.

Suppose g has such a real-valued solution z(x). Since we are dealing only with real-valued functions the term $(z^{(n)})^2$ forces $z^{(n)} = 0$ and thus z must be a polynomial of degree (n-1) at most. Each w_i was so chosen that if $z = a_{n-1}x^{n-1} + a_0$, then $w_i = a_{i+1}$. Since

$\sin \pi w_i = 0$, a_i is forced to be an integer. Moreover, since $Q(y, w_1, w_2, \ldots, 2^{w_1}) = 0$, Q must possess a set of integer solutions $w_i = a_{i+1}$.

Suppose $Q(y, x_1, \ldots, x_n, 2^{x_1}) = 0$ has solutions $x_i = a_i$, a_i integers. Then $z(x) = a_{n-1}x^{n-1} + \ldots + a_0$ is a solution to g=0.

The statement of Theorem 2 is too general to make it a satisfying decision problem since the set of all real-valued functions with real derivatives is not computable. The theorem would hold for any computable superset of functions of the set of polynomials of degree n with integer coefficients.

Theorem 2 seems to indicate the concept of a real-valued solution to a differential equation is quite elusive.

APPENDIX C

SIN'S PERFORMANCE ON SAINT'S PROBLEMS

As an experiment for testing SIN's performance, we attempted the 86 problems attempted by SAINT and reported in Slagle's thesis. SAINT integrated 84 our of these 86 problems and announced failure to integrate $x\sqrt{1+x}$ and $\cos\sqrt{x}$. Slagle reports that SAINT solved the 84 problems with an average time of 2.4 minutes (144 seconds). SIN solved all 86 problems with an average time of 2.4 seconds. This average becomes 1.3 seconds when one discounts the cost of chaining. Chaining occurred on 22 our of the 86 problems. Chaining is considered to take 4.5 seconds in this accounting. That time appears to be a minimum bound for the operation. In order to determine the time required by SIN to solve a problem, we used the execution time reported by CTSS. The swap time in CTSS is ignored here.

Over half of the 86 problems (more precisely 45) were completely solved by SIN's first stage. These problems were solved with an average time of 0.6 seconds. Of the remaining problems only two required the Integration-by-parts routine (i.e., $x \cos x$ and $\cos\sqrt{x}$ - the latter generates the subproblem $\int 2y \cos y \, dy$). Two routines were added to SIN in order to solve the definite and double integrals among the 86 problems. These routines call SIN to perform the integrations indicated and make appropriate substitutions at the upper and lower bounds.

Below we list problems for which SAINT results are available and the comparative results for SIN.

Problem	SAINT time in seconds	SIN time in seconds	discount for chain	Notes
$\int_1^2 \frac{1}{x}\,dx$	1.8	0.20		Fastest problem solved by SAINT, integrated by table look up in IMSLN
$\int \frac{\sec^2 t\,dt}{1 + \sec^2 t - 3\tan t}$	1080	9.18	4.6	Longest solution time in SAINT. 9 subgoals in SAINT, 1 in SIN
$\int \frac{dx}{\sec^2 x}$	126	0.87		7 subgoals in SAINT, 3 in SIN
$\int \frac{x^2 + 1}{\sqrt{x}}\,dx$	102	5.87	1.3	3 subgoals SAINT 1 SIN
$\int \frac{x}{\sqrt{x^2 + 2x + 5}}\,dx$	960	9.68	5.2	14 subgoals SAINT 1 SIN
$\int \sin^2 x \cos x\,dx$	120	0.33		
$\int (\sin^2 x + 1)^2 \cos x\,dx$	228	2.48		
$\int \frac{e^x dx}{1 + e^x}$	102	0.28		2 subgoals SAINT 0 SIN
$\int \frac{e^{2x}}{1 + e^x}\,dx$	222	6.23	1.7	
$\int \frac{1}{1 - \cos x}\,dx$	120	9.78	5.3	
$\int_0^{\frac{\pi}{3}} \tan x \sec^2 x\,dx$	144	0.47		

Problem	SAINT time in seconds	SIN time in seconds	discount for chain	Notes
$\int_0^1 x \log_e x \, dx$	132	0.70		
$\int_0^{\frac{\pi}{6}} \sin x \cos x \, dx$	156	0.30		Largest speed ratio between SIN and SAINT
$\int \frac{x + 1}{\sqrt{2x - x^2}} \, dx$	576	10.1	5.6	Longest solution in SIN. 13 subgoals SAINT 1 SIN
$\int \frac{2e^x}{2 + 3e^{2x}} \, dx$	360	8.25	3.7	4 subgoals SAINT 1 SIN
$\int \frac{x^4}{(1 - x^2)^{5/2}} \, dx$	660	8.77	4.3	13 subgoals SAINT 2 SIN
$\int \frac{e^{6x}}{e^{4x} + 1} \, dx$	510	7.92	3.5	10 subgoals SAINT 1 SIN
$\int \log_e(2 + 3x^2) dx$	390	7.20	2.7	10 subgoals SAINT 1 SIN

The last 3 problems were solved by SAINT in 540, 318 and 210 seconds respectively after an entry was added to SAINT's table which was used in the solution of these problems.

In order to fully account for the effect of garbage collection the problems were run in large batches. Thus garbage collection time was distributed over the set of problems. Garbage collection time probably accounts for less than 20% of the total time in SIN.

We should note some of the reasons for the time difference in the results of SAINT and SIN. SAINT was run on the 7090 and SIN on the 7094. This accounts for about 40% of the gain (2.18 vs. 2.00 microseconds in the cycle time and overlapped instruction execution in the 7094). The single major difference in the time is due to the fact that SAINT ran mostly interpreted (a major exception being the simplifier), and SIN was run mostly compiled. Compilation is usually considered to gain a factor of 20-30 in the speed of the program. We tested some problems with SIN being executed completely interpretively. We noted an average speed loss of a factor of 15. However none of the problems which were run interpretively included problems which required chaining. Thus we were unable to run some of the more complex problems in the set interpretively.

By taking these factors into account we note that SIN would only run about three times faster than SAINT on the average when both are executed under optimal conditions. The reason for the relatively small ratio in SIN's favor we believe is because most of the time spent in SIN in solving the harder problems in the set is spent in algebraic manipulations (e.g., simplifications). Algebraic manipulation in SIN is not materially faster than it is in SAINT. Though the analysis performed in SIN yields a very direct solution, the total time spent to obtain the solution is still significant. Hence the contrast with SAINT in regard to total solution time is not very great.

APPENDIX D

Solution of Problems Proposed by McIntosh

Professor McIntosh (National Poleytechnic Institute of Mexico) required the solution of eleven nontrivial integration problems for a physics paper that he was writing [35]. He found the solution to these problems in Petit Bois' table. He also asked us to solve these problems using SIN. The problems involved variable coefficients in a square root of a quadratic which the version of SIN current at that time was not equipped to handle. Although we had intended to add the variable coefficient capability to Method 5, it was not needed for the SAINT experiment described in Appendix C. We rewrote Method 5 to account for variable coefficients. Interestingly enough this was not sufficient for a satisfactory solution of the problems since Professor McIntosh required that the output be in terms of the arcsin function. In some cases the transformations proposed by Method 5 yielded an answer in terms of the log function. To force the arcsin result a further method was added. Thus if the integral was of the form

$$\int \frac{C}{x\sqrt{ax^2+bx+c}}\,dx$$

the substitution $y = \frac{1}{x}$ was made. This substitution rids the denominator of the factor x. With these modifications SIN was able to solve all eleven problems. In the solutions obtained by McIntosh

we noted some discrepancies from solutions obtained by SIN. It should be noted, however, that McIntosh was only interested in the coefficient of the arcsin terms and not in the argument. All the errors were minor and occurred only in the arguments of the arcsin function.

Important lessons are to be obtained from this experiment. It is quite likely that other users of SIN will have similar requirements regarding the form of the output. SIN should therefore be modified so that FORM can accept simple descriptions of new substitutions written, say, as a SCHATCHEN and REPLACE rule.

An examination of the eleven problems will indicate that a great deal of SIN's machinery was involved in solving these problems. Thus it would appear that a program such as SIN is more useful than a special purpose integration routine written for solving just this set of problems. Such a special purpose program will require so much machinery as to make it uneconomical.

Finally we should note that this experiment points out the need for further work on methods which transform algebraic integrands. The method we introduced to force the arcsin result also decreased the labor involved in the solution and should be normally available in SIN.

McIntosh Problems

	Problem	Constraints	Answer equivalent to
1)	$\int \frac{dr}{r\sqrt{2Hr^2 - \alpha^2}}$	$H > 0$	$-\frac{1}{\alpha} \arcsin \frac{\alpha}{\sqrt{2H}\, r}$
2)	$\int \frac{dr}{r\sqrt{2Hr^2 - \alpha^2 - \epsilon^2}}$	$H > 0$	$\frac{-1}{\sqrt{\alpha^2 + \epsilon^2}} \arcsin \frac{\sqrt{\alpha^2 + \epsilon^2}}{\sqrt{2H}\, r}$
3)	$\int \frac{dr}{r\sqrt{2Hr^2 - \alpha^2 - 2Kr^4}}$	$H^2 > 2\alpha^2 K$	$\frac{1}{2\alpha} \arcsin \frac{Hr^2 - \alpha^2}{r^2\sqrt{H^2 - 2K\alpha^2}}$
4)	$\int \frac{dr}{r\sqrt{2He^2 - \alpha^2 - \epsilon^2 - 2Kr^4}}$	$H^2 > 2(\alpha^2 + \epsilon^2)\, K$	$\frac{1}{2\sqrt{\alpha^2 + \epsilon^2}} \arcsin \frac{Hr^2 - (\alpha^2 + \epsilon^2)}{r^2\sqrt{H^2 - 2(\alpha^2+\epsilon^2)K}}$
5)	$\int \frac{dr}{r\sqrt{2Hr^2 - \alpha^2 - 2Kr}}$	$K^2 + 2H\alpha^2 > 0$	$\frac{1}{\alpha} \arcsin \frac{Kr - \alpha^2}{r\sqrt{K^2 + 2H\alpha^2}}$
6)	$\int \frac{dr}{r\sqrt{2Hr^2 - \alpha^2 - \epsilon^2 - 2Kr}}$	$K^2 + 2(\alpha^2 + \epsilon^2)H > 0$	$\frac{1}{\sqrt{\alpha^2 + \epsilon^2}} \arcsin \frac{-Kr - (\alpha^2 + \epsilon^2)}{r\sqrt{K^2 + 2(\alpha^2+\epsilon^2)H}}$
7)	$\int \frac{r\, dr}{\sqrt{2Er^2 - \alpha^2}}$		$\frac{1}{2E}\sqrt{2Er^2 - \alpha^2}$
8)	$\int \frac{r\, dr}{\sqrt{2Er^2 - \alpha^2 - \epsilon^2}}$		$\frac{1}{2E}\sqrt{2Er^2 - (\alpha^2 + \epsilon^2)}$

	Problems	Constraints	Answer equivalent to
9)	$\int \frac{r\,dr}{\sqrt{2Er^2 + \alpha^2 - 2Kr^4}}$	$E^2 > 2K\alpha^2$	$\frac{1}{2\sqrt{2K}} \arcsin \frac{2K\,r^2 - E}{\sqrt{E^2 - 2K\alpha^2}}$
10)	$\int \frac{r\,dr}{\sqrt{2Er^2 - \alpha^2 - \epsilon^2 - 2Kr^4}}$	$E^2 > 2K(\alpha^2 + \epsilon^2)$ $K > 0$	$\frac{1}{2\sqrt{2K}} \arcsin \frac{2Kr^2 - E}{\sqrt{E^2 - 2K(\alpha^2 + \epsilon^2)}}$
11)	$\int \frac{r\,dr}{\sqrt{2Er^2 - \alpha^2 - 2Kr}}$	$E < 0$	$= \frac{\sqrt{2Er^2 - \alpha^2 - 2Kr}}{2E} + \frac{1}{2HE\sqrt{-2E}} \arcsin \frac{2Er + K}{\sqrt{K^2 - 2E\alpha^2}}$

APPENDIX E

AN EXPERIMENT WITH SOLDIER

As an experiment for testing the effectiveness of the differential equations routines we attempted to solve the review problems appearing in pages 54-56 of "Applied Differential Equations" by Spiegel [60]. This text was chosen for sentimental reasons since it was the book through which we first learned methods for solving ordinary differential equations. The methods described in Chapter 6 were mostly influenced by Ince's "Integration of Ordinary Differential Equations" [27], and Kamke's "Differentialgleichungen" [30]. As it turns out the methods in Spiegel were quite similar, which is not a surprising fact. However, there were some differences and these will be pointed out below.

Briefly, the results of the experiment were as follows: Of the 80 problems in pages 54-56 of the book, 4 involved second and higher order equations (i.e., y", y'''). These problems were not attempted since SOLDIER had no machinery to deal with them. Thus the number of problems actually attempted was 76. Of the 76, SOLDIER satisfactorily solved 67 problems with an average time of 6.6 seconds. Discounting the cost incurred by chaining (chaining occurred on 26 of these 66 problems), the average time was 4.3 seconds. Two problems were completely reduced to integration problems, but were not integrated by

problems were not solved at all. An examination of the result reported by SOLDIER for one of the problems (i.e., 51) indicated a misprint in the book. As before, our timing information is based on the report by CTSS of the execution time of the program.

The system on which this experiment was carried out had the following characteristics: SCHATCHEN, SCHVUOS, FORM, REPLACE, SOLDIER, and all the solution methods for differential equations were compiled. A few integration methods, especially the Derivative-divides method, were also compiled. The rest of the integration methods were run interpretively. This accounted for a noticeable increase in solution time when one of the integration subproblems required a solution method in stage 2 or 3 of SIN. As was the case in the experiment reported in Appendix C, the 76 problems were attempted in large batches (about 15 at a time) so that the effects due to garbage collection were fully considered.

Below we shall describe on the performance of SOLDIER on some of the more interesting fully solved problems. We shall then describe each of the 9 problems which it failed to solve fully.

Representative Solved Problems

The largest number of integrations needed to solve one of the 67 problems was 3. This was achieved by problem 69 among others.

(69) $(e^y+x+3)y' = 1$ or $(e^y+x+3)dy - dx = 0$

This problem is solved by one of the multiplier methods (Chapter 6, Method 3)

$\frac{\partial}{\partial x}(e^y+x+3) = 1$

$\frac{\partial}{\partial y}(-1) = 0$

$-\frac{1}{1}(1-0) = -1$, and -1 is a function of y.

Thus the first integral is

$\int -1\, dy = -y$

The multiplier is e^{-y} resulting in the exact equation

$(1+xe^{-y}+3e^{-y})dy - e^{-y}dx = 0$

The second integral is

$\int -e^{-y}dx = -xe^{-y}$,

and the final integral is

$$\int (1+3e^{-y})dy = y-3e^{-y}$$

The solution reported by SOLDIER is thus

$$Co = -xe^{-y}-3e^{-y}+y$$

The solution in Spiegel is

$$x = ye^{y}-3+ce^{y}.$$

This solution is equivalent to the one obtained by SOLDIER. This problem was solved in 5.2 seconds.

The most complex solution was obtained as a result to problem 73.

(73) $$\frac{dy}{dx} = \frac{x+3y}{x-3y}$$

This homogeneous problem required the solution of

$$\int \frac{du}{u-\frac{1+3u}{1-3u}}$$

The final solution given by SOLDIER was

$$\log_e x + \tfrac{1}{2}\log_e\left(1+3\frac{y^2}{x^2} + 2\frac{y}{x}\right) - \sqrt{2}\ \arctan\left(\frac{1}{\sqrt{2}} + \frac{3y}{\sqrt{2}x}\right) = Co$$

The solution in Spiegel was

$$\log_e(x^2+2xy+3y^2) = 2\sqrt{2}\ \arctan\left(\frac{x+3y}{x\sqrt{2}}\right) + c$$

This problem was solved in 15.3 seconds and required a chain to the rational function package.

The problem in which we discovered a misprint in the book's solution was problem 51.

(51) $\quad y' = 3x+2y \qquad$ or $y'-3x-2y = 0$

The problem is linear (Chapter 6, Method 1) and the first integral required is

$$\int -2dx = -2x$$

The next integral is

$$\int -3x\, e^{-2x}dx = \left(\frac{3}{4} + \frac{3}{2}x\right)e^{-2x}$$

The final answer given by SOLDIER was

$$Co = ye^{-2x} + \left(\frac{3}{4} + \frac{3}{2}x\right)e^{-2x}$$

The book's solution was

$$y = c\, e^{-2x} - \frac{3}{2}x - \frac{3}{4}$$

This solution differed from SOLDIER's in that the sign of the exponent of e^{-2x} is wrong.

The answer was obtained in 9.0 seconds and required a chain to solve the second integral.

The fastest solution time was obtained for problem 5.

(5) $(3-y)dx + 2xdy = 0, \quad y(1) = 1$

This problem is also linear.

The first integral is

$$\int -\frac{1}{2x}\, dx = -\frac{1}{2} \log_e x$$

The next integral (after simplifying $e^{-1/2 \log_e x} = \frac{1}{\sqrt{x}}$) is

$$\int \frac{3}{2x^{3/2}}\, dx = -\frac{3}{\sqrt{x}}$$

The final result is

$$Co = \frac{y}{\sqrt{x}} - \frac{3}{\sqrt{x}}$$

The book's solution is

$$(3-y)^2 = 4x$$

which is equivalent ot SOLDIER's except that the constant of integration was determined by using the initial condition.

This problem was solved in 0.8 seconds.

The Nine Unsolved Problems

Problems 48 and 75 were not solved primarily because SOLDIER had no machinery for factoring them. In these two

(48) $$\frac{dq}{dp} = \frac{p}{q} e^{p^2-q^2}$$

(75) $$e^{2x-y}dy + e^{y-2x}dy = 0$$

problems what is needed is to recognize that $e^{a+b} = e^a e^b$. A powerful factoring routine would have yielded the result that both of these problems are separable.

Problem 50 is also recognized to be separable

(50) $$(x+x\cos y)dy - (y+\sin y)dx = 0$$

if one factors x+xcosy. When SOLDIER solved this problem it utilized one of the multiplier methods.

The difficulties due to the lack of a general factoring or division routine which was pointed out in Chapter 6 is one of the outstanding problems which must be solved in order to achieve a powerful routine for solving differential equations. The rational function package which is not directly utilized by SOLDIER can factor polynomials and some more general expressions (e.g., x+xcosy

could be factored by it), however, it must be extended in order to recognize factorizations involving exponentials and logs.

A similar difficulty to factoring faced the program in problem 65.

(65) $$xy' + y\log_e x = y\log_e y + y$$

This problem is easily solved by the homogeneous method if it is first transformed into

$$xy' - y\log_e \frac{y}{x} = y$$

SOLDIER does not possess enough machinery to realize that this transformation can be effected. Method 8 of Chapter 6 which normally would have solved problem 65 without the log transformation failed because SCHVUOS could not simplify a quotient which arose in the course of the solution.

Problems 47 and 64 were not solved because SOLDIER lacked a method given in Spiegel.

(47) $$xdy - ydx = x^2ydy$$

(64) $$xdy - ydx = 2x^2y^2dy$$

Spiegel suggested that one should watch out for frequently occurring combinations such as xdy+ydx or xdy-ydx. He gave a method which deals with some of these cases. In 47 he points out that by

dividing by x^2 one obtains the derivative of $\frac{y}{x}$ on the left hand side and ydy on the right hand side. In 64 one obtains $2y^2dy$ on the right hand side and once again the derivative of $\frac{y}{x}$ on the left hand side. SOLDIER lacked this particular method and was unable to solve these problems. Once again Method 8 of Chapter 6 was applicable and did not find a solution due to problems in division.

Another method lacking in the program is pointed out by problem 57.

(57) $$\frac{ds}{dt} = \frac{1}{s+t+1}$$

Here the linear substitution u(t) = s+t+1 would have left a separable equation. Also a reversal of the independent variable followed by multiplying out the denominator would have left the equation

$$\frac{dt}{ds} = s+t+1$$

which is linear. The method of multiplying out the denominator is also useful in problem 17.

(17) $$y' = \frac{2xy-y^4}{3x^2}$$

SOLDIER solved 17 by dividing through the denominator and using the Bernoulli method. By multiplying out the denominator, the multiplier

method would solve the problem.

Problem 22 was not solved by SOLDIER because the almost-linear method is not powerful enough.

(22) $$(\tan y - \tan^2 y \cos x)dx - x\sec^2 y\, dy = 0$$

The substitution $u(x)=\tan(y)$ results in the equation

$$(u-u^2\cos x)dx - x\, du = 0$$

which is Bernoulli. However, the almost-linear method checks only for the possibility that the resulting equation is linear and completely misses the possibility that it is Bernoulli.

Finally, two problems, 56 and 74, were not completely solved because SIN did not have powerful enough machinery.

(56) $$\frac{dI}{dt} + 3I = 10\sin t$$

(74) $$y'\cos x = y - \sin 2x$$

In 56 the linear method generates the subproblem

$$\int -10e^{3t}\sin t\, dt$$

Without the Edge heuristic, SIN cannot integrate this problem. There was not enough room in the system to include the Edge heuristic (only 1500 words were left in free storage), so SIN failed to

integrate this problem.

SIN failed to handle the integration problems needed in 74 because it does not currently possess enough machinery for dealing with sin(2x) and cos(x) in the same integrand. As has been indicated in Chapter 4 some machinery for just this situation was designed but not fully implemented.

Modifications to SOLDIER

Following the experiment reported above we made two changes to the methods employed by SOLDIER. First we added a simple factorization routine to Method 8 of Chapter 6. With this routine Method 8 was able to solve problems 47, 64, and 65, as expected.

In addition we added an indicator to SCHVUOS. When this indicator was on, SCHVUOS executed the rule $e^{a+b} \rightarrow e^a e^b$. This indicator was turned on in running Method 2 of Chapter 6 (Separable). Thus, problems 48 and 75 were solved as well. The use of indicators illustrates the approach toward simplification programs we had outlined in Chapter 3. In that chapter we said that simplifiers should be considered as black boxes with strings attached. When a decision has to be made inside the simplification program, it can check to see whether it had been given an instruction regarding the choice to be made.

These changes must be considered as stop-gap measures and not as solutions to the factoring problems which still remain in SOLDIER.

APPENDIX F

LISTINGS

The listings of SIN and SOLDIER given below were produced by a LISP program written by Diffie of the MATHLAB project and modified by us. Listings of LISP programs are frequently printed by using the internal representation of the program. The listings of programs written in most other languages usually bear a close correspondence to the input form of the program. This need not be the case for LISP programs. The routine Edge which was not listed using Diffie's program is presented last. The listing of this routine may be used to guage the effect of Diffie's program.

The listings of two recent LISP programs (i.e., Martin [37], Norton [47]) are also available. One can use these listings to compare different styles of LISP programming. Norton accentuates the use of the PROG feature and his programs thus have a FORTRAN-like appearance. Martin's style is richer and leans toward greater use of "pure" LISP. Our style is intermediate to these two styles.

SCHATCHEN

```
DEFINE
(((SCHATCHEN M2)
  (M2 (LAMBDA (E P SPLIST)
              (PROG (ANS)
                    (RETURN (COND ((NULL (M1 E P)) NIL)
                                  ((NULL ANS) T)
                                  (T ANS) )))))
  (M1 (LAMBDA (E P)
              (COND ((EQUAL E P) T)
                    ((ATOM P) NIL)
                    ((ATOM (CAR P))
                     (COND ((OR (EQ (CAR P) (QUOTE PLUS))
                                (EQ (CAR P) (QUOTE TIMES)) )
                            (LOOPP E P) )
                           ((EQ (CAR P) (QUOTE EXPT)) (ZEPOW E P))
                           ((EQ (CAR E) (CAR P)) (EACHP E P))
                           ((OP (CAR P)) NIL)
                           ((EQ (CAR P) (QUOTE COEFFT))
                            (COEFFPORT E P (QUOTE (TIMES 1 T))) )
                           ((EQ (CAR P) (QUOTE COEFFPT)) (COEFFPT E P T))
                           ((EQ (CAR P) (QUOTE COEFFP))
                            (COEFFPORT E P (QUOTE (PLUS 0 T))) )
                           ((EQ (CAR P) (QUOTE COEFFTT))
                            (COEFFTT E (CADR P) T (QUOTE TIMES)) )
                           ((EQ (CAR P) (QUOTE COEFFPP))
                            (COEFFTT E (CADR P) T (QUOTE PLUS)) )
                           ((EQ (CAR P) (QUOTE DVCOE)) (DVCOE E P T))
                           ((EQ (CAR P) (QUOTE ZEPOW)) (ZEPOW E P))
                           ((AND (SETQ ANS (CONS NIL ANS)) (TESTA P E NIL))
                            (RESTORE1) )
                           (T (RESTORE)) ))
                    ((ATOM (CAAR P))
                     (COND ((ATOM E) NIL)
                           ((PROG2 (SETQ ANS (CONS NIL ANS))
                                   (TESTA (CAR P) (CAR E) E) )
                            (COND ((OR (EQ (CAR E) (QUOTE PLUS))
                                       (EQ (CAR E) (QUOTE TIMES)) )
                                   (COND ((LOOPP E
                                                 (CONS (CAR E)
                                                       (CDR P) ))
                                          (RESTORE1) )
                                         (T (RESTORE)) ))
                                  ((AND (SETQ P (CONS (CAR E) (CDR P)))
                                        (EACHP E P) )
                                   (RESTORE1) )
                                  (T (RESTORE)) ))
                           (T (RESTORE)) ))
                     (T NIL) )))))
DEFINE
(((LOOPP (LAMBDA (E P)
                 (PROG (X Z EE)
                       (SETQ EE
                             (COND ((NOT (EQ (CAR E) (CAR P)))
                                    (LIST (CAR P) E) )
```

```
                (T E) ))
    (SETQ Z P)
    (SETQ ANS (CONS NIL ANS))
LOOP
    (SETQ Z (CDR Z))
    (COND ((NULL Z)
           (RETURN (COND ((NULL (CDR EE)) (RESTORE1))
                         (T (RESTORE)) ))))
    (SETQ X EE)
L5
    (COND ((NULL (CDR X)) (GO L17))
          ((OP1 (CAAR Z)) (GO L10))
          ((EQ (CAAR Z) (QUOTE EXPT)) (GO L14))
          ((M1 (CADR X) (CAR Z)) (GO L2)) )
L8
    (SETQ X (CDR X))
    (GO L5)
L2
    (SETQ ANS (CONS (CONS X (CDR X)) ANS))
    (RPLACD X (CDDR X))
    (GO LOOP)
L17
    (COND ((NOT (EQ (CAR P) (QUOTE PLUS))) (GO L18))
          ((M1 0 (CAR Z)) (GO LOOP)) )
L19
    (RETURN (RESTORE))
L18
    (COND ((AND (EQ (CAR P) (QUOTE TIMES))
                (M1 1 (CAR Z)) )
           (GO LOOP) )
          (T (RETURN (RESTORE))) )
L10
    (COND ((EQ (CAAR Z) (QUOTE COEFFT)) (GO L11))
          ((EQ (CAAR Z) (QUOTE COEFFP)) (GO L12))
          ((EQ (CAAR Z) (QUOTE COEFFPT)) (GO L13))
          ((EQ (CAAR Z) (QUOTE COEFFTT)) (GO L16))
          ((EQ (CAAR Z) (QUOTE COEFFPP)) (GO L47))
          ((EQ (CAAR Z) (QUOTE ZEPOW)) (GO L14))
          ((EQ (CAAR Z) (QUOTE DVCOE)) (GO L43))
          (T (GO L15)) )
L11
    (COND ((COEFFPORT EE (CAR Z) (QUOTE (TIMES 1 NIL)))
           (GO LOOP) )
          (T (RETURN (RESTORE))) )
L12
    (COND ((COEFFPORT EE (CAR Z) (QUOTE (PLUS 0 NIL)))
           (GO LOOP) )
          (T (RETURN (RESTORE))) )
L13
    (COND ((COEFFPT EE (CAR Z) NIL) (GO LOOP))
          (T (RETURN (RESTORE))) )
L14
    (COND ((ZEPOW (CADR X) (CAR Z)) (GO L2)) (T (GO L8)))
L15
    (COND ((LOOP EE (CDAR Z)) (GO LOOP))
          (T (RETURN (RESTORE))) )
L16
    (COND ((COEFFTT EE (CADAR Z) NIL (QUOTE TIMES))
           (GO LOOP) )
          (T (RETURN (RESTORE))) )
L47
```

```
                    (COND ((COEFFTT EE (CADAR Z) NIL (QUOTE PLUS))
                           (GO LOOP) )
                          (T (RETURN (RESTORE))) )
               L43
                    (COND ((DVCOE (CADR X) (CAR Z) NIL) (GO LOOP))
                          (T (GO L8)) ))))))
DEFINE
(((COEFFPORT
   (LAMBDA
    (E P IND)
    (PROG (X Z EE)
          (SETQ ANS (CONS NIL ANS))
          (SETQ EE E)
          (COND
           ((EQ (CAR IND) (QUOTE PLUS)) (GO L30))
           ((EQ (CAR E) (QUOTE PLUS)) (GO L31))
           ((EQ (CAR E) (QUOTE TIMES)) (GO L32)) )
          (SETQ EE (LIST (QUOTE TIMES) E))
          (GO L2)
     L32
          (COND ((CADDR IND) (GO L2)) (T (GO L1)))
     L31
          (COND
           ((NOT (CADDR IND)) (GO L1))
           ((NULL (CDDR E)) (GO L2))
           (T (GO L20)) )
     L30
          (COND ((EQ (CAR E) (QUOTE PLUS)) (GO L35)))
          (SETQ EE (LIST (QUOTE PLUS) E))
          (GO L2)
     L35
          (COND
           ((NULL (CDDR E)) (GO L2))
           ((EQ (CAR IND) (QUOTE PLUS)) (GO L2))
           ((CADDR IND) (GO L2))
           (T (GO L1)) )
     L2
          (COND ((EQUAL E 0) (GO L7)))
          (SETQ Z (CDR P))
     LOOP1
          (SETQ Z (CDR Z))
          (COND ((NULL Z) (GO L7)))
          (SETQ X EE)
     L6
          (COND
           ((NULL (CDR X)) (GO L10))
           ((EQ (CAAR Z) (QUOTE COEFFTT)) (GO L16))
           ((EQ (CAAR Z) (QUOTE COEFFPP)) (GO L17))
           ((M1 (CADR X) (CAR Z)) (GO L5)) )
          (SETQ X (CDR X))
          (GO L6)
     L5
          (SETQ ANS (CONS (CONS X (CDR X)) ANS))
          (RPLACD X (CDDR X))
          (GO LOOP1)
     L17
          (COND ((COEFFTT EE (CADAR Z) NIL (QUOTE PLUS)) (GO LOOP1)))
          (GO L7)
     L16
          (COND ((COEFFTT EE (CADAR Z) NIL (QUOTE TIMES)) (GO LOOP1)))
     L7
```

```
          (COND
           ((NULL (CDR EE))
            (RETURN (COND ((TESTA (CADR P) (CADR IND) NIL)
                           (COND ((CADDR IND) (RESTORE1)) (T (RESTORE2))) )
                          (T (RESTORE)) )))
           ((NULL (CDDR EE))
            (RETURN (COND ((TESTA (CADR P) (CADR EE) NIL)
                           (PROG2 (SETQ ANS
                                        (CONS (CONS EE (CDR EE)) ANS) )
                                  (PROG2 (RPLACD EE (CDDR EE))
                                         (COND ((CADDR IND)
                                                (RESTORE1) )
                                               (T (RESTORE2)) ))))
                          (T (RESTORE)) ))))
     L69
          (SETQ X (COPY1 EE))
          (COND ((NULL (TESTA (CADR P) X NIL)) (RETURN (RESTORE)))
           ((CADDR IND) (RETURN (RESTORE1))) )
          (COND ((AND (CDDR E) (EQ (CAR IND) (QUOTE PLUS)))
            (PROG2 (SETQ ANS (CONS (CONS EE (CDR EE)) ANS)) (RPLACD EE NIL)) ))
          (RETURN (RESTORE2))
     L10
          (COND ((NULL (M1 (CADR IND) (CAR Z))) (RETURN (RESTORE))))
          (GO LOOP1)
     L20
          (RETURN (RESTORE))
     L1
          (SETQ X EE)
     L3
          (COND ((NULL (CDR X)) (GO L4))
           ((COEFFPORT (CADR X) P (LIST (CAR IND) (CADR IND) T)) (GO L12)) )
          (SETQ X (CDR X))
          (GO L3)
     L12
          (SETQ ANS (CONS (CONS X (CDR X)) ANS))
          (RPLACD X (CDDR X))
          (RETURN (RESTORE2))
     L4
          (COND ((NULL (M1 (CADR IND) P)) (RETURN (RESTORE))))
          (RETURN (RESTORE2)) )))))
DEFINE
(((COEFFPT (LAMBDA (E P IND)
                   (PROG (Z ZZ)
                         (SETQ Z
                               (COND ((EQ (CAR E) (QUOTE PLUS)) E)
                                     (T (LIST (QUOTE PLUS) E)) ))
                         (SETQ ANS (CONS NIL ANS))
                         (SETQ ZZ (CONS (QUOTE COEFFT) (CDR P)))
                    L19
                         (COND ((NULL (CDR Z)) (GO L21))
                               ((NULL (M1 (CADR Z) ZZ)) (GO L20)) )
                    L22
                         (SETQ ANS (CONS (CONS Z (CDR Z)) ANS))
                         (RPLACD Z (CDDR Z))
                         (GO L19)
                    L20
                         (SETQ Z (CDR Z))
                         (GO L19)
                    L21
                         (SETQ Z
                               (FINDIT (COND ((EQ (CAADR P) (QUOTE VAR*))
```

```
                                        (CAR (CDDADR P)) )
                                       (T (CAADR P)) )))
                    (COND ((NULL Z)
                           (RETURN (COND ((NULL (TESTA (CADR P)
                                                       0
                                                       NIL ))
                                          (RESTORE) )
                                         ((IND (RESTORE1))
                                         (T (PROG2 (RESTORE2) 0)) )))
                          ((NULL (CDR Z))
                           (RETURN (COND ((NULL (TESTA (CADR P)
                                                       (CAR Z)
                                                       NIL ))
                                          (RESTORE) )
                                         ((IND (RESTORE1))
                                         (T (PROG2 (RESTORE2)
                                                   (CAR Z) ))))))
                    (SETQ Z (SIMPPLUS Z))
                    (COND ((NULL (TESTA (CADR P) Z (QUOTE COEFFPT)))
                           (RETURN (RESTORE)) )
                          ((IND (RETURN (RESTORE1))) )
                    (RETURN (PROG2 (RESTORE2) Z)) )))
(EACHP (LAMBDA (E P)
               (PROG NIL
                     (COND ((NOT (EQUAL (LENGTH E) (LENGTH P)))
                            (RETURN NIL) ))
                     (SETQ ANS (CONS NIL ANS))
                EACHPL
                     (SETQ E (CDR E))
                     (COND ((NULL E) (RETURN (RESTORE1)))
                           ((NULL (M1 (CAR E) (CADR P)))
                            (RETURN (RESTORE)) ))
                     (SETQ P (CDR P))
                     (GO EACHPL) )))
(ZEPOW (LAMBDA (E P)
               (PROG NIL
                     (SETQ ANS (CONS NIL ANS))
                     (COND ((ATOM E) (GO L6)))
                L5
                     (COND ((NOT (EQ (CAR E) (QUOTE EXPT))) (GO L8))
                           ((NOT (M1 (CADR E) (CADR P))) (GO L8))
                           ((NOT (M1 (CADDR E) (CADDR P)))
                            (RETURN (RESTORE)) ))
                L9
                     (RETURN (RESTORE1))
                L10
                     (COND ((AND (NOT (M1 0 (CADDR P)))
                                 (NOT (M1 1 (CADR P))) )
                            (RETURN (RESTORE)) ))
                     (GO L9)
                L8
                     (COND ((NOT (M1 E (CADR P))) (RETURN (RESTORE)))
                           ((NOT (M1 1 (CADDR P))) (RETURN (RESTORE))) )
                     (GO L9)
                L7
                     (COND ((NOT (M1 0 (CADR P))) (RETURN (RESTORE))))
                     (GO L9)
                L6
                     (COND ((EQP E 1) (GO L10))
                           ((EQP E 0) (GO L7))
                           (T (GO L8)) ))))
```

```
  (LOOP (LAMBDA (E LP)
                (PROG (Z Y X)
                      (SETQ ANS (CONS (QUOTE *LOOP) (CONS NIL ANS)))
                      (SETQ X LP)
                 L5
                      (SETQ Z E)
                 L6
                      (COND ((NULL (M1 (CADR Z) (CAR X))) (GO L10)))
                      (SETQ Y (CONS (LIST X Z (CDR Z)) Y))
                      (SETQ ANS (CONS (CONS Z (CDR Z)) ANS))
                      (RPLACD Z (CDDR Z))
                      (SETQ X (CDR X))
                      (COND ((NULL X) (RETURN (RESTORE2))))
                      (SETQ ANS (CONS (QUOTE *LOOP) ANS))
                      (GO L5)
                 L10
                      (SETQ Z (CDR Z))
                      (COND ((NOT (NULL (CDR Z))) (GO L6))
                            ((EQUAL X LP) (RETURN (RESTORE))) )
                 L8
                      (SETQ X (CAAR Y))
                      (RPLACD (CADAR Y) (CADDAR Y))
                      (SETQ Z (CADDAR Y))
                      (SETQ Y (CDR Y))
                      (SETQ ANS (CDR ANS))
                      (RESTORE3)
                      (GO L6) ))))
DEFINE
(((RESTORE3 (LAMBDA NIL
                    (PROG NIL
                     L1
                          (COND ((NULL ANS) (ERROR (QUOTE RESTORE3)))
                                ((NULL (CAR ANS)) (ERROR (QUOTE RESTORE3)))
                                ((EQ (CAR ANS) (QUOTE *LOOP)) (RETURN NIL))
                                ((NOT (ATOM (CAAR ANS)))
                                 (RPLACD (CAAR ANS) (CDAR ANS)) ))
                          (SETQ ANS (CDR ANS))
                          (GO L1) )))
  (RESTORE (LAMBDA NIL
                   (PROG (Y)
                         (SETQ Y ANS)
                    L1
                         (COND ((NULL Y) (RETURN NIL))
                               ((EQ (CAR Y) (QUOTE *LOOP))
                                (PROG2 (RPLACA Y (CADR Y))
                                       (RPLACD Y (CDDR Y)) ))
                               ((NULL (CAR Y))
                                (RETURN (PROG2 (SETQ ANS (CDR Y)) NIL)) )
                               ((NOT (ATOM (CAAR Y)))
                                (RPLACD (CAAR Y) (CDAR Y)) ))
                         (SETQ Y (CDR Y))
                         (GO L1) )))
  (RESTORE1 (LAMBDA NIL
                    (PROG (Y)
                     L2
                          (SETQ Y ANS)
                          (COND ((NULL ANS) (RETURN T))
                                ((NULL (CAR ANS))
                                 (RETURN (PROG2 (SETQ ANS (CDR ANS)) T)) )
                                ((NOT (ATOM (CAAR ANS))) (GO L3)) )
                     L1
```

```
                        (COND ((NULL (CDR Y)) (RETURN T))
                              ((NULL (CADR Y))
                               (RETURN (PROG2 (RPLACD Y (CDDR Y)) T)) )
                              ((NOT (ATOM (CAADR Y)))
                               (PROG2 (RPLACD (CAADR Y) (CDADR Y))
                                      (RPLACD Y (CDDR Y)) ))
                              (T (SETQ Y (CDR Y))) )
                        (GO L1)
                   L3
                        (RPLACD (CAAR ANS) (CDAR ANS))
                        (SETQ ANS (CDR ANS))
                        (GO L2) )))
  (RESTORE2 (LAMBDA NIL
                   (PROG (Y)
                        (SETQ Y ANS)
                        (COND ((NULL ANS) (RETURN T))
                              ((NULL (CAR ANS))
                               (RETURN (PROG2 (SETQ ANS (CDR ANS)) T)) ))
                   L1
                        (COND ((NULL (CDR Y)) (RETURN T))
                              ((EQ (CADR Y) (QUOTE *LOOP))
                               (RPLACD Y (CDDR Y)) )
                              ((NULL (CADR Y))
                               (RETURN (PROG2 (RPLACD Y (CDDR Y)) T)) ))
                        (SETQ Y (CDR Y))
                        (GO L1) )))
  (TESTA* (LAMBDA (ALA EXP LOC)
                  (COND ((COND ((EQ (CADR ALA) (QUOTE FREE)) (FREE EXP))
                               ((EQ (CADR ALA) (QUOTE NUMBERP))
                                (NUMBERP EXP) )
                               ((EQ (CADR ALA) (QUOTE TRUE)) T)
                               (T (APPLY (CADR ALA)
                                         (FINDTHEM (CDDR ALA))
                                         (ALIST) )))
                         (COND ((NOT (MEMBER (CAR ALA) SPLIST))
                                (PROG2 (SETQ ANS
                                             (CONS (CONS (CAR ALA) EXP)
                                                   ANS ))
                                       T ))
                               (T T) ))
                        (T NIL) )))
  (FINDTHEM (LAMBDA (ARGS) (FINDT1 ARGS ANS (CONS EXP NIL))))
  (FINDT1 (LAMBDA (X Y Z)
                  (COND ((NULL X) Z)
                        ((NULL Y)
                         (FINDT1 (CDR X)
                                 ANS
                                 (NCONC Z (LIST (EVAL (CAR X) (ALIST))))) ))
                        ((EQ (CAAR Y) (CAR X))
                         (FINDT1 (CDR X) ANS (NCONC Z (CONS (CDAR Y) NIL))) )
                        (T (FINDT1 X (CDR Y) Z)) )))))
DEFINE
(((OP (LAMBDA (FN)
              (MEMBER FN
                      (QUOTE (PLUS TIMES
                                   EXPT
                                   SIN
                                   COS
                                   TAN
                                   LOG
                                   SEC
```

```
                                INTEGRAL
                                ARCSIN
                                ARCCOS
                                ARCTAN )))))
(COPY1 (LAMBDA (A) (COND ((NULL A) NIL) (T (CONS (CAR A) (COPY1 (CDR A)))))))
(FINDIT (LAMBDA (A)
                (PROG (Y Z)
                      (SETQ Y (CONS NIL ANS))
                 L1
                      (COND ((NULL (CDR Y)) (RETURN Z))
                            ((NULL (CADR Y)) (RETURN Z))
                            ((EQ (CAADR Y) A)
                             (PROG2 (SETQ Z (NCONC Z (LIST (CDADR Y))))
                                    (RPLACD Y (CDDR Y)) ))
                            (T (SETQ Y (CDR Y))) )
                      (GO L1) )))
(FREE (LAMBDA (A)
              (COND ((ATOM A) (NOT (EQ A VAR)))
                    (T (AND (FREE (CAR A)) (FREE (CDR A)))) )))
(OP1 (LAMBDA (A)
             (MEMBER A
                     (QUOTE (COEFFPT COEFFP
                                     COEFFT
                                     ZEPOW
                                     COEFFPP
                                     COEFFTT
                                     LOOP )))))
(COEFFTT (LAMBDA (EXP PAT IND OPIND)
                 (PROG (RES Z)
                       (SETQ ANS (CONS NIL ANS))
                       (COND ((AND IND (NOT (EQ (CAR EXP) OPIND)))
                              (SETQ EXP (LIST OPIND EXP)) ))
                       (SETQ Z EXP)
                       (SETQ SPLIST (CONS (CAR PAT) SPLIST))
                  L1
                       (COND ((NULL (CDR Z)) (GO L3))
                             ((TESTA PAT (CADR Z) NIL) (GO L2)) )
                       (SETQ Z (CDR Z))
                       (GO L1)
                  L2
                       (SETQ ANS (CONS (CONS Z (CDR Z)) ANS))
                       (SETQ RES (CONS (CADR Z) RES))
                       (RPLACD Z (CDDR Z))
                       (GO L1)
                  L3
                       (SETQ SPLIST (CDR SPLIST))
                       (COND (RES (GO L4))
                             ((NOT (TESTA PAT
                                          (COND ((EQ OPIND
                                                     (QUOTE PLUS) )
                                                 0 )
                                                (T 1) )
                                          NIL ))
                              (RETURN (RESTORE)) ))
                       (COND (IND (RETURN (RESTORE1)))
                             (T (RETURN (RESTORE2))) )
                  L4
                       (SETQ RES
                             (COND ((CDR RES) (CONS OPIND RES))
                                   (T (CAR RES)) ))
                       (SETQ ANS (CONS (CONS (CAR PAT) (SIMP RES)) ANS))
```

```
                        (COND (IND (RETURN (RESTORE1)))
                              (T (RETURN (RESTORE2))) ))))
  (TESTA
   (LAMBDA (ALA EXP B)
           (PROG (Y Z FUNC VAL)
                 (COND ((NOT (EQ (CAR ALA) (QUOTE VAR*)))
                        (RETURN (TESTA* ALA EXP NIL)) ))
                 (SETQ Z (CADR ALA))
                 (SETQ ALA (CDDR ALA))
            LOOP
                 (COND ((NULL Z)
                        (RETURN (PROG2 (SETQ Y
                                             (COND (VAL (M1 EXP Y))
                                                   (T (TESTA* ALA
                                                              EXP
                                                              NIL ))))
                                       (COND ((NULL Y) NIL)
                                             (FUNC (SET (CAR ALA) EXP))
                                             (T Y) ))))
                       ((EQ (CAR Z) (QUOTE SET)) (SETQ FUNC T))
                       ((EQ (CAR Z) (QUOTE UVAR))
                        (COND ((SETQ Y
                                     (CDR (SASSOC (CAR ALA)
                                                  ANS
                                                  (QUOTE NILL) )))
                               (SETQ VAL T) )
                              (T NIL) ))
                       ((AND (EQ B (QUOTE COEFFPT))
                             (EQ (CAAR Z) (QUOTE COEFFPT)) )
                        (SETQ ALA (CADAR Z)) ))
                 (SETQ Z (CDR Z))
                 (GO LOOP) )))))


            SCHVUOS, REPLACE, DIFF


DEFINE
(((SIMPPLUS
   (LAMBDA
    (EXP)
    (PROG (Y IND Z W ANS A B A1)
          (SETQ A 0)
     B
          (COND ((NULL EXP) (GO AA)))
          (SETQ Y (SIMP (CAR EXP)))
          (COND
           ((EQ (CAR Y) (QUOTE PLUS)) (GO C))
           ((NUMBERP Y) (SETQ A (PLUS Y A)))
           (T (SETQ Z (CONS Y Z))) )
     BB
          (SETQ EXP (CDR EXP))
          (GO B)
     C
          (COND
           ((NUMBERP (CADR Y))
            (PROG2 (SETQ Z (APPEND (CDDR Y) Z)) (SETQ A (PLUS (CADR Y) A))) )
           (T (SETQ Z (APPEND (CDR Y) Z))) )
          (GO BB)
     AA
          (COND
```

```
          ((NULL Z) (GO E1))
          ((NULL (CDR Z)) (GO EE))
          ((EQ (CAAR Z) (QUOTE TIMES)) (GO E)) )
     H
          (SETQ A1 1)
          (SETQ IND T)
          (SETQ B (CAR Z))
          (GO FF)
     E
          (COND ((NOT (NUMBERP (CADAR Z))) (GO P)))
          (SETQ A1 (CADAR Z))
          (COND ((NULL (CDDDAR Z)) (GO G)))
          (SETQ B (CDDAR Z))
          (SETQ IND NIL)
          (GO FF)
     P
          (SETQ A1 1)
          (SETQ B (CDAR Z))
          (SETQ IND NIL)
          (GO FF)
     G
          (SETQ B (CADDAR Z))
          (SETQ IND T)
     FF
          (SETQ Z (CONS (QUOTE PLUS) (CDR Z)))
          (SETQ Y
           (COND
            (IND (COEFFPT Z (LIST NIL (QUOTE (C NUMBERP)) B) NIL))
            (T (COEFFPT Z (CONS NIL (CONS (QUOTE (C NUMBERP)) B)) NIL)) ))
          (SETQ Y (PLUS A1 Y))
          (COND
           ((ZEROP Y) T)
           ((ONEP Y)
            (SETQ W (CONS (COND (IND B) (T (CONS (QUOTE TIMES) B))) W)) )
           (IND (SETQ W (CONS (LIST (QUOTE TIMES) Y B) W)))
           (T (SETQ W (CONS (CONS (QUOTE TIMES) (CONS Y B)) W))) )
          (SETQ Z (CDR Z))
          (GO AA)
     EE
          (SETQ W (CONS (CAR Z) W))
     E1
          (SETQ W (COND ((ZEROP A) W) (T (CONS A W))))
          (RESTORE)
          (COND
           ((NULL W) (RETURN 0))
           ((NULL (CDR W)) (RETURN (CAR W)))
           (T (RETURN (CONS (QUOTE PLUS) W))) ))))))

DEFINE
(((SIMPTIMES
   (LAMBDA
    (EXP)
    (PROG (Y DIV Z W A A1 B ZZ)
          (SETQ A 1)
     B
          (COND ((NULL EXP) (GO START)))
          (SETQ Y (SIMP (CAR EXP)))
          (COND ((EQ (CAR Y) (QUOTE TIMES))
                 (COND ((NUMBERP (CADR Y))
                        (PROG2 (SETQ A (TIMES (CADR Y) A))
                               (SETQ Z (APPEND (CDDR Y) Z)) ))
```

```
                (T (SETQ Z (APPEND (CDR Y) Z))) ))
          ((AND (NUMBERP Y) (ZEROP Y)) (RETURN 0))
          ((NUMBERP Y) (SETQ A (TIMES Y A)))
          (T (SETQ Z (CONS Y Z))) )
     (SETQ EXP (CDR EXP))
     (GO B)
START
     (COND ((AND (EQ (CAAR Z) (QUOTE PLUS))
                 (NULL (CDR Z))
                 (NULL W)
                 (NOT (ONEP A)) )
            (RETURN (PROG23 (CSETQ SIMPIND T)
                            (TIMESLOOP A (CDAR Z))
                            (CSETQ SIMPIND NIL) ))))
     (COND ((NULL Z) (GO E1))
           ((NULL (CDR Z)) (GO EE))
           (EXPTSUM (RETURN (CONS (QUOTE TIMES) (CONS A Z))))
           ((EQ (CAAR Z) (QUOTE EXPT)) (GO G)) )
     (SETQ A1 1)
     (SETQ B (CAR Z))
     (GO FF)
G
     (SETQ B (CADAR Z))
     (SETQ A1
           (COND ((NUMBERP (CADDAR Z)) (CADDAR Z))
                 (T (CONS (CADDAR Z) NIL)) ))
FF
     (SETQ ZZ Z)
K
     (COND ((EQ (CAADR ZZ) (QUOTE EXPT)) (GO H))
           ((M2 (CADR ZZ) B NIL) (GO I)) )
     (COND ((AND QUOTIND
                 (EQ (CAR B) (QUOTE PLUS))
                 (EQ (CAADR ZZ) (QUOTE PLUS))
                 (SETQ Y (MATCHSUM1 B (CADR ZZ))) )
            (GO DIV1) ))
JK
     (SETQ ZZ (CDR ZZ))
J
     (COND ((CDR ZZ) (GO K)))
     (GO M)
H
     (COND ((M2 (CADADR ZZ) B NIL) (GO L)))
     (COND ((AND QUOTIND
                 (EQ (CAR B) (QUOTE PLUS))
                 (EQ (CAR (CADADR ZZ)) (QUOTE PLUS))
                 (SETQ Y (MATCHSUM1 B (CADADR ZZ))) )
            (GO DIV2) ))
     (GO JK)
JJ
     (RPLACD ZZ (CDDR ZZ))
     (GO J)
I
     (SETQ A1 (COND ((NUMBERP A1) (ADD1 A1)) (T (CONS 1 A1))))
     (GO JJ)
L
     (SETQ A1
           (COND ((AND (NUMBERP A1) (NUMBERP (CADDAR (CDR ZZ))))
                  (PLUS A1 (CADDAR (CDR ZZ))) )
                 (T (CONS (CADDAR (CDR ZZ))
                          (COND ((ATOM A1) (LIST A1)) (T A1)) ))))
```

```
        (GO JJ)
    M
        (SETQ A1 (COND ((NUMBERP A1) A1) (T (SIMPPLUS A1))))
        (SETQ W
              (COND ((NUMBERP A1)
                     (COND ((ZEROP A1) W)
                           ((ONEP A1) (CONS B W))
                           (T (CONS (LIST (QUOTE EXPT) B A1) W)) ))
                    (T (CONS (LIST (QUOTE EXPT) B A1) W)) ))
        (SETQ Z (CDR Z))
        (GO START)
    EE
        (SETQ W (CONS (CAR Z) W))
    E1
        (SETQ A
              (COND ((NULL W) A)
                    ((NULL (CDR W))
                     (COND ((ONEP A) (CAR W))
                           (T (LIST (QUOTE TIMES) A (CAR W))) ))
                    ((ONEP A) (CONS (QUOTE TIMES) W))
                    (T (CONS (QUOTE TIMES) (CONS A W))) ))
        (COND ((NULL DIV) (RETURN A))
              (T (RETURN (SIMPTIMES (LIST (CONS (QUOTE TIMES) DIV) A)))) )
    DIV1
        (COND ((AND (NUMBERP Y) (SETQ A (TIMES A Y))) (GO I))
              ((SETQ DIV (CONS Y DIV)) (GO I)) )
    DIV2
        (SETQ DIV (CONS (SIMPEXPT (LIST Y (CAR (CDDADR ZZ)))) DIV))
        (GO L) )))))

DEFINE
(((SIMPEXPT
   (LAMBDA
    (EXP)
    (PROG (A B)
          (SETQ B (SIMP (CADR EXP)))
          (SETQ A (SIMP (CAR EXP)))
          (COND
           ((EQP A 0) (RETURN 0))
           ((AND
             (EQ (CAR A) (QUOTE EXPT))
             (SETQ B (SIMPTIMES (LIST B (CADDR A))))
             (SETQ A (CADR A))
             NIL )
            NIL )
           ((EQP B 0) (RETURN 1))
           ((EQP B 1) (RETURN A))
           ((EQP A 1) (RETURN 1))
           ((AND (NUMBERP A) (NUMBERP B))
            (RETURN (COND
              ((NOT EXPTIND) (EXPT A B))
              ((AND (FIXP B) (GREATERP B -1)) (EXPT A B))
              (T (LIST (QUOTE EXPT) A B)) )))
           ((EQ (CAR A) (QUOTE TIMES))
            (RETURN (CONS (QUOTE TIMES) (EXPTLOOP (CDR A)))) )
           ((AND EXPTSUM (EQ (CAR B) (QUOTE PLUS)))
            (RETURN
             (CONS
              (QUOTE TIMES)
              (MAPLIST (CDR B)
               (FUNCTION (LAMBDA (C) (SIMPEXPT (LIST A (CAR C))))) ))))
```

```
          ((NOT (ATOM B))
           (RETURN
            (PROG (W)
                  (RETURN
                   (COND
                    ((NOT (SETQ W
                               (M2
                                B
                                (QUOTE (PLUS (COEFFT (C TRUE1)
                                                     (LOG (B1 TRUE) (A TRUE)) )
                                             (COEFFP (E TRUE)) ))
                                NIL )))
                     (LIST (QUOTE EXPT) A B) )
                    ((NOT (EQUAL A (SUBLIS W (QUOTE B1))))
                     (LIST (QUOTE EXPT) A B) )
                    (T
                     (SIMPTIMES (LIST
                       (SIMPEXPT (LIST (SUBLIS W (QUOTE A))
                         (SUBLIS W (QUOTE C)) ))
                       (SIMPEXPT (LIST A (SUBLIS W (QUOTE E)))) )))))))))
          (RETURN (LIST (QUOTE EXPT) A B)) )))
  (EXPTLOOP
   (LAMBDA
    (A)
    (PROG23
     (CSETQ SIMPIND T)
     (MAPLIST A (FUNCTION (LAMBDA (C) (SIMPEXPT (LIST (CAR C) B)))))
     (CSETQ SIMPIND NIL) )))
  (SIMP
   (LAMBDA
    (EXP)
    (PROG (Z)
          (RETURN
           (COND
            ((ATOM EXP) EXP)
            (SIMPIND EXP)
            ((NULL (SETQ Z (GET (CAR EXP) (QUOTE SIMP))))
             (CONS (CAR EXP)
               (MAPLIST (CDR EXP) (FUNCTION (LAMBDA (C) (SIMP (CAR C)))))) )
            ((EQ Z (QUOTE SIMPTIMES)) (SIMPTIMES (CDR EXP)))
            ((EQ Z (QUOTE SIMPPLUS)) (SIMPPLUS (CDR EXP)))
            ((EQ Z (QUOTE SIMPEXPT)) (SIMPEXPT (CDR EXP)))
            (T (APPLY Z (LIST (CDR EXP)) (ALIST))) ))))))

ATTRIB
(PLUS (SIMP SIMPPLUS))

ATTRIB
(TIMES (SIMP SIMPTIMES))

ATTRIB
(EXPT (SIMP SIMPEXPT))

DEFINE
(((SIMPLOG
   (LAMBDA
    (A)
    (PROG (B)
          (SETQ B (SIMP (CADR A)))
          (SETQ A (SIMP (CAR A)))
          (COND ((EQUAL A B) (RETURN 1))
```

```
                ((EQP B 1) (RETURN 0))
                ((EQ (CAR B) (QUOTE EXPT))
                 (COND ((EQUAL A (CADR B)) (RETURN (CADDR B)))
                       (T (RETURN (LIST (QUOTE TIMES)
                                        (CADDR B)
                                        (LIST (QUOTE LOG) A (CADR B)) )))))
                (T (RETURN (LIST (QUOTE LOG) A B))) ))))))

ATTRIB
(LOG (SIMP SIMPLOG))

DEFINE
(((SIMPTRIG
   (LAMBDA
    (A B C D)
    (PROG (Y)
          (RETURN (COND
            ((EQUAL D B) C)
            ((ATOM D) (LIST A D))
            ((SETQ Y
                   (CDR (SASSOC (CAR D)
                                (GET A (QUOTE SIMPTRIG))
                                (QUOTE NILL) )))
             (SIMP (SUBST (CADR D) (QUOTE X) Y)) )
            (T (LIST A D)) )))))
  (SIMPTRIG1 (LAMBDA (A) (SIMPTRIG (QUOTE SIN) 0 0 (SIMP (CAR A))))) ))

ATTRIB
(SIN (SIMP SIMPTRIG1))

ATTRIB
(COS (SIMP SIMPTRIG2))

DEFINE
(((SIMPTRIG2 (LAMBDA (A) (SIMPTRIG (QUOTE COS) 0 1 (SIMP (CAR A)))))))

DEFINE
(((TIMESLOOP
   (LAMBDA
    (A B)
    (CONS
     (QUOTE PLUS)
     (MAPLIST B
      (FUNCTION (LAMBDA (C)
        (SIMPTIMES (PROG23 (CSETQ SIMPIND T) (LIST A (CAR C)) (CSETQ SIMPIND NIL)) ))))))))
  (EXPAND
   (LAMBDA
    (A B)
    (SIMPPLUS (MAPLIST B (FUNCTION (LAMBDA (C) (TIMESLOOP (CAR C) A))))) ))
  (PROG23 (LAMBDA (A B C) B)) ))

DEFINE
(((SIMPTAN (LAMBDA (A)
                   (COND ((EQ (CAAR A) (QUOTE ARCTAN)) (SIMP (CADAR A)))
                         (T (SIMPTRIG (QUOTE TAN) 0 0 (SIMP (CAR A)))) )))
  (SIMPARCTAN (LAMBDA (A)
                      (COND ((EQ (CAAR A) (QUOTE TAN)) (SIMP (CADAR A)))
                            (T (SIMPTRIG (QUOTE ARCTAN) 0 0 (SIMP (CAR A)))) )))))

ATTRIB
(TAN (SIMP SIMPTAN))
```

```
ATTRIB
(ARCTAN (SIMP SIMPARCTAN))

DEFINE
(((SIMPDIFFERENCE (LAMBDA (A)
                          (SIMPPLUS (LIST (CAR A)
                                          (SIMPTIMES (LIST -1 (CADR A))) ))))
  (SIMPQUOTIENT (LAMBDA (A)
                        (SIMPTIMES (LIST (CAR A)
                                         (SIMPEXPT (LIST (CADR A) -1)) ))))
  (SIMPMINUS (LAMBDA (A) (SIMPTIMES (LIST -1 (CAR A))))) ))

ATTRIB
(DIFFERENCE (SIMP SIMPDIFFERENCE))

ATTRIB
(QUOTIENT (SIMP SIMPQUOTIENT))

ATTRIB
(MINUS (SIMP SIMPMINUS))

ATTRIB
(SIN (SIMPTRIG ((ARCSIN . X)
                (ARCCOS EXPT (DIFFERENCE 1 (EXPT X 2)) 0.5E0)
                (ARCTAN QUOTIENT X (EXPT (PLUS 1 (EXPT X 2)) 0.5E0)) )))

ATTRIB
(COS (SIMPTRIG ((ARCSIN EXPT (DIFFERENCE 1 (EXPT X 2)) 0.5E0)
                (ARCCOS . X)
                (ARCTAN EXPT (PLUS 1 (EXPT X 2)) -0.5E0) )))

ATTRIB
(TAN (SIMPTRIG ((ARCSIN QUOTIENT X (EXPT (DIFFERENCE 1 (EXPT X 2)) 0.5E0))
                (ARCCOS QUOTIENT (EXPT (DIFFERENCE 1 (EXPT X 2)) 0.5E0) X)
                (ARCTAN . X) )))

ATTRIB
(ARCSIN (SIMPTRIG ((SIN . X) (COS PLUS X (QUOTIENT PI 2)))))

ATTRIB
(ARCCOS (SIMPTRIG ((SIN DIFFERENCE X (QUOTIENT PI 2)) (COS . X))))

ATTRIB
(ARCTAN (SIMPTRIG ((TAN . X))))

DEFINE
(((NILL (LAMBDA NIL (QUOTE (NIL))))))

DEFINE
(((SIMPARCSIN (LAMBDA (A) (SIMPTRIG (QUOTE ARCSIN) 0 0 (SIMP (CAR A)))))
  (SIMPARCCOS
   (LAMBDA (A)
           (SIMPDIFFERENCE (LIST (SIMPQUOTIENT (LIST (QUOTE PI) 2))
                                 (SIMPARCSIN (LIST A)) ))))
  (SIMPARCCOT
   (LAMBDA (A)
           (SIMPDIFFERENCE (LIST (SIMPQUOTIENT (LIST (QUOTE PI) 2))
                                 (SIMPARCTAN (LIST A)) ))))))
```

```
ATTRIB
(ARCSIN (SIMP SIMPARCSIN))

ATTRIB
(ARCCOS (SIMP SIMPARCCOS))

ATTRIB
(ARCCOT (SIMP SIMPARCCOT))

DEFINE
(((MATCHSUM1 (LAMBDA (ASUM BSUM)
                     (PROG (Z W LENGTH MINLENGTH QUOT MINQUOT)
                           (COND ((NOT (EQUAL (LENGTH ASUM) (LENGTH BSUM)))
                                  (RETURN NIL) ))
                           (SETQ Z (CADR ASUM))
                           (SETQ W (CDR BSUM))
                           (SETQ MINLENGTH 1000)
                     LOOP
                           (SETQ QUOT (SIMPQUOTIENT (LIST (CAR W) Z)))
                           (SETQ LENGTH
                                 (LENGTH (COND ((EQ (CAR QUOT)
                                                    (QUOTE TIMES) )
                                                (CDR QUOT) )
                                               (T (QUOTE (NIL))) )))
                           (COND ((GREATERP LENGTH MINLENGTH) (GO A)))
                           (SETQ MINLENGTH LENGTH)
                           (SETQ MINQUOT QUOT)
                     A
                           (COND ((EQUAL MINLENGTH 1) (GO OUT)))
                           (SETQ W (CDR W))
                           (COND (W (GO LOOP)))
                     OUT
                           (COND ((M2 BSUM
                                      (TIMESLOOP MINQUOT (CDR ASUM))
                                      NIL )
                                  (RETURN MINQUOT) ))
                           (RETURN NIL) )))))

DEFINE
(((SIMPCOT (LAMBDA (X) (LIST (QUOTE EXPT) (SIMPTAN X) -1)))))

ATTRIB
(COT (SIMP SIMPCOT))

DEFINE
(((REPLACE (LAMBDA (DICT EXP1)
    (PROG23 (CSETQ SIMPIND T) (REPLAC EXP1) (CSETQ SIMPIND NIL)) ))
  (REPLAC
   (LAMBDA
    (EXP1)
    (PROG (Z1)
          (RETURN
           (COND
            ((NULL EXP1) NIL)
            ((NOT (ATOM EXP1))
             (COND
              ((EQ (CAR EXP1) (QUOTE EVAL))
               (PROG2
                (SETQ Z1 (EVAL (REPLAC (CADR EXP1)) (ALIST)))
                (PROG23
                 (CSETQ SIMPIND NIL)
```

```
                (SIMP Z1)
                (CSETQ SIMPIND T) )))
             ((EQ (CAR EXP1) (QUOTE QUOTE*)) (CADR EXP1))
             (T (PROG (Z1 W1)
                      (SETQ Z1 (REPLAC (CAR EXP1)))
                      (SETQ W1 (REPLAC (CDR EXP1)))
                      (RETURN (COND ((AND (EQ Z1 (CAR EXP1)) (EQ W1 (CDR EXP1)))
                                 EXP1 )
                                (T (SIMP1 (CONS Z1 W1))) ))))))
          ((NUMBERP EXP1) EXP1)
          ((SETQ Z1 (SASSOC EXP1 DICT (FUNCTION (LAMBDA NIL NIL))))
           (CDR Z1) )
          (T EXP1) )))))
  (SIMP1 (LAMBDA (EXP1)
    (COND
     ((ATOM EXP1) EXP1)
     ((NOT (GET (CAR EXP1) (QUOTE SIMP))) EXP1)
     ((EQ (CAR EXP1) (QUOTE TIMES)) (SIMPTIMES (CDR EXP1)))
     ((EQ (CAR EXP1) (QUOTE PLUS)) (SIMPPLUS (CDR EXP1)))
     ((EQ (CAR EXP1) (QUOTE EXPT)) (SIMPEXPT (CDR EXP1)))
     (T (APPLY (GET (CAR EXP1) (QUOTE SIMP)) (LIST (CDR EXP1)) (ALIST))) )))))

DEFINE
(((DVCOE
   (LAMBDA (E P IND)
           (PROG (X Y Z)
                 (SETQ ANS (CONS NIL ANS))
                 (COND ((NOT (EQ (CAR E) (QUOTE TIMES)))
                        (SETQ E (LIST (QUOTE TIMES) E)) ))
                 (SETQ Z (CDR P))
            LOOP
                 (SETQ Z (CDR Z))
                 (COND ((NULL Z)
                        (COND ((TESTA (CADR P) (SIMP (COPY1 E)) NIL)
                               (RETURN (COND (IND (RESTORE1))
                                             (T (RESTORE2)) )))
                              (T (RETURN (RESTORE))) )))
                    (SETQ X E)
                 (GO LOOP2)
            LOOP1
                 (SETQ X (CDR X))
            LOOP2
                 (COND ((NULL (CDR X)) (GO L6)))
                 (COND ((EQ (CAADR X) (QUOTE EXPT)) (GO L1))
                       ((M1 (CADR X) (CAR Z)) (GO L2)) )
                 (GO LOOP1)
            L2
                 (SETQ ANS (CONS (CONS X (CDR X)) ANS))
                 (RPLACD X (CDDR X))
                 (GO LOOP)
            L1
                 (COND ((EQ (CAAR Z) (QUOTE EXPT)) (GO L3))
                       ((NOT (M1 (CADADR X) (CAR Z))) (GO LOOP1)) )
                 (SETQ Y -1)
            L7
                 (SETQ ANS (CONS (CONS X (CDR X)) ANS))
                 (RPLACD X
                         (CONS (SIMP (LIST (CAADR X)
                                           (CADADR X)
                                           (LIST (QUOTE PLUS)
                                                 (CAR (CDDADR X))
```

```
                                                    Y )))
                                  (CDDR X) ))
                  (GO LOOP)
             L3
                  (COND ((M1 (CADADR X) (CADAR Z)) (GO L5)))
                  (GO LOOP1)
             L5
                  (COND ((M1 (CAR (CDDADR X)) (CADDAR Z)) (GO L2)))
                  (SETQ Y (SIMPMINUS (LIST (CADDAR Z))))
                  (GO L7)
             L6
                  (COND ((M1 1 (CAR Z)) (GO LOOP)))
                  (SETQ E
                        (CONS (CAR E)
                              (CONS (SIMPEXPT (LIST (CAR Z) -1)) (CDR E)) ))
                  (GO LOOP) )))))

DEFINE
(((DIFF1 (LAMBDA (EXP VAR) (PROG23 (CSET SIMPIND T) (DIFF EXP) (CSET SIMPIND NIL))) )
  (DIFF
   (LAMBDA
    (EXP)
    (COND
     ((ATOM EXP) (COND ((EQ EXP VAR) 1) (T 0)))
     ((EQ (CAR EXP) (QUOTE EXPT))
      (COND
       ((FREE (CADDR EXP))
        (SIMPTIMES (LIST
          (CADDR EXP)
          (SIMPEXPT (LIST (CADR EXP) (SIMPPLUS (LIST (CADDR EXP) -1))))
          (DIFF (CADR EXP)) )))
       ((FREE (CADR EXP))
        (SIMPTIMES (LIST
          EXP
          (SIMPLOG (LIST (QUOTE E) (CADR EXP)))
          (DIFF (CADDR EXP)) )))
       (T
        (SIMPTIMES
         (LIST
          EXP
          (SIMPPLUS (LIST
            (SIMPTIMES (LIST
              (CADDR EXP)
              (DIFF (CADR EXP))
              (SIMPEXPT (LIST (CADR EXP) -1)) ))
            (SIMPTIMES (LIST (SIMPLOG (LIST (QUOTE E) (CADR EXP)))
              (DIFF (CADDR EXP)) ))))))))))
     ((EQ (CAR EXP) (QUOTE TIMES))
      (SIMPPLUS
       (MAPLIST
        (CDR EXP)
        (FUNCTION (LAMBDA (Y)
          (SIMPTIMES (CONS (DIFF (CAR Y)) (CHOICE (CAR Y) (CDR EXP)))) ))))))
     ((EQ (CAR EXP) (QUOTE PLUS))
      (SIMPPLUS (MAPLIST (CDR EXP) (FUNCTION (LAMBDA (Y) (DIFF (CAR Y)))))) )
     (T (APPLY (GET (CAR EXP) (QUOTE DIFF)) (LIST (CDR EXP)) (ALIST))) )))
  (CHOICE (LAMBDA (A B)
    (COND ((EQ A (CAR B)) (CDR B)) (T (CONS (CAR B) (CHOICE A (CDR B))))) ))))

DEFINE
(((BIGDIFF (LAMBDA (A B)
```

```
                (SIMPTIMES (LIST (DIFF (CAR A))
                                 (SUBST (CAR A) (QUOTE X) B) ))))))

DEFINE
(((DIFLOG (LAMBDA (A)
    (PROG NIL (SETQ A (CDR A)) (RETURN (BIGDIFF A (QUOTE (EXPT X -1))))) ))
  (DIFSIN (LAMBDA (A) (BIGDIFF A (QUOTE (COS X)))))
  (DIFCOS (LAMBDA (A) (BIGDIFF A (QUOTE (TIMES -1 (SIN X))))))
  (DIFTAN (LAMBDA (A) (BIGDIFF A (QUOTE (EXPT (SEC X) 2)))))
  (DIFSEC (LAMBDA (A) (BIGDIFF A (QUOTE (TIMES (SEC X) (TAN X))))))
  (DIFARCTAN (LAMBDA (A) (BIGDIFF A (QUOTE (EXPT (PLUS 1 (EXPT X 2)) -1))))))
  (DIFARCSIN (LAMBDA (A)
    (BIGDIFF A (QUOTE (EXPT (PLUS 1 (TIMES -1 (EXPT X 2))) -0.5E0))) ))
  (DIFCSC (LAMBDA (A) (BIGDIFF A (QUOTE (TIMES -1 (COT X) (CSC X))))))
  (DIFCOT (LAMBDA (A) (BIGDIFF A (QUOTE (TIMES -1 (EXPT (CSC X) 2))))))
  (DIFARCCOS (LAMBDA (A) (MINUS (DIFARCSIN A))))
  (DIFARCSEC
   (LAMBDA (A)
           (BIGDIFF A
                    (QUOTE (EXPT (TIMES X
                                        (EXPT (DIFFERENCE (EXPT X 2) 1)
                                              0.5E0 ))
                                 -1 ))))))
  (DIFARCCSC (LAMBDA (A) (SIMPMINUS (LIST (DIFARCSEC A)))))
  (DIFINTEGRAL (LAMBDA (X)
    (COND ((EQ (CADR X) VAR) (CAR X))
      (T (SIMP (LIST (QUOTE INTEGRAL) (DIFF (CAR X)) (CADR X)))) ))))))

ATTRIB
(INTEGRAL (DIFF DIFINTEGRAL))

ATTRIB
(SIN (DIFF DIFSIN))

ATTRIB
(COS (DIFF DIFCOS))

ATTRIB
(TAN (DIFF DIFTAN))

ATTRIB
(SEC (DIFF DIFSEC))

ATTRIB
(ARCTAN (DIFF DIFARCTAN))

ATTRIB
(ARCSIN (DIFF DIFARCSIN))

ATTRIB
(LOG (DIFF DIFLOG))

ATTRIB
(CSC (DIFF DIFCSC))

ATTRIB
(COT (DIFF DIFCOT))

ATTRIB
(ARCCOS (DIFF DIFARCCOS))
```

```
ATTRIB
(ARCSEC (DIFF DIFARCSEC))

ATTRIB
(ARCCSC (DIFF DIFARCCSC))

DEFINE
(((EXPAND2 (LAMBDA (EXP) (PROG23 (CSET SIMPIND T) (EXPAND1 EXP) (CSET SIMPIND NIL)))
)
  (EXPAND1
   (LAMBDA
    (EXP)
    (COND
     ((ATOM EXP) EXP)
     ((AND (EQ (CAR EXP) (QUOTE EXPT))
           (NOT (ATOM (CADR EXP)))
           (INTEGERP (CADDR EXP))
           (EQ (CAADR EXP) (QUOTE PLUS))
           (GREATERP (CADDR EXP) 0)
           (LESSP (CADDR EXP) 6) )
      (EXPANDEXPT (CADR EXP) (CADDR EXP)) )
     ((EQ (CAR EXP) (QUOTE TIMES))
      (COND ((CDDR EXP)
             (PRODEXPAND (EXPAND1 (CADR EXP))
                         (EXPAND1 (CONS (QUOTE TIMES) (CDDR EXP))) ))
            ((CDR EXP) (EXPAND1 (CADR EXP)))
            (T NIL) ))
     (T (SIMP1 (MAPLIST EXP (FUNCTION (LAMBDA (C) (EXPAND1 (CAR C))))))) )))
  (PRODEXPAND (LAMBDA (A B)
    (COND
     ((NOT (OR (EQ (CAR A) (QUOTE PLUS)) (EQ (CAR B) (QUOTE PLUS))))
      (SIMPTIMES (LIST A B)) )
     ((NOT (EQ (CAR A) (QUOTE PLUS))) (TIMESLOOP A (CDR B)))
     ((NOT (EQ (CAR B) (QUOTE PLUS))) (TIMESLOOP B (CDR A)))
     (T (EXPAND (CDR A) (CDR B))) )))))

DEFINE
(((RATIONALIZE
   (LAMBDA
    (EXP)
    (PROG (W)
          (RETURN
           (COND
            ((NOT (EQ (CAR EXP) (QUOTE PLUS))) NIL)
            ((SETQ
              W
              (M2
               EXP
               (QUOTE
                (PLUS
                 (TIMES
                  (COEFFTT
                   (C
                    (FUNCTION
                     (LAMBDA
                      (C)
                      (M1
                       C
                       (QUOTE
                        (EXPT
                         (AA (FUNCTION (LAMBDA (AA)
```

```
                              (AND (NOT (EQUAL AA 1))
                                   (NOT (EQUAL AA 0)) ))))
                          (N (FUNCTION (LAMBDA (N)
                                   (AND (NUMBERP N) (LESSP N 0)) ))))))))))
                    (COEFFTT (B TRUE)) )
                   (COEFFPT (A TRUE)) ))
                NIL ))
             (REPLACE W (QUOTE (TIMES (PLUS (QUOTIENT A C) B) C))) )
            (T NIL) )))))))


            FORM,SIN,DERIVATIVE-DIVIDES


DEFINE
(((TRUE1 (LAMBDA (A) (OR (NOT (NUMBERP A)) (NOT (ZEROP A)))))
  (INTEGERP1 (LAMBDA (A) (INTEGERP (SIMPTIMES (LIST 2 A)))))
  (VARP (LAMBDA (A) (EQUAL A VAR)))
  (FREE1 (LAMBDA (A) (AND (FREE A) (OR (NOT (NUMBERP A)) (NOT (ZEROP A))))))
  (FIXP1 (LAMBDA (A) (AND (NUMBERP A) (FIXP A))))
  (MASTER (LAMBDA (A)
    (PROG NIL
          (FILEWRITE (QUOTE MANOVE) (QUOTE LISP) (QUOTE MASTER))
          (FILEAPND
           (QUOTE MANOVE)
           (QUOTE LISP)
           (LIST (CONS (CAR A) (TRANSL (SIMP (CDR A))))) )
          (CHAIN (QUOTE ((SAVE MOSES T) (R FULMAN MANOVE))))
          (FILESEEK (QUOTE MANOVE) (QUOTE ANS))
          (RETURN (SIMP (UNTR (READ)))) )))))

DEFINE
(((FORM
   (LAMBDA
    (EXPRES)
    (COND
     ((FREE EXPRES) NIL)
     ((ATOM EXPRES) NIL)
     ((MEMBER (CAR EXPRES) (QUOTE (PLUS TIMES)))
      ((LAMBDA (L)
         (PROG (Y)
          LOOP
               (COND
                ((SETQ Y (FORM (CAR L))) (RETURN Y))
                ((NOT (SETQ L (CDR L))) (RETURN NIL))
                (T (GO LOOP)) )))
       (CDR EXPRES) ))
     ((MEMBER (CAR EXPRES) (QUOTE (LOG ARCTAN ARCSIN)))
      (COND
       ((SETQ ARG
          (M2
           EXP
           (LIST
            (QUOTE TIMES)
            (QUOTE (COEFFTT (C RAT8PRIME)))
            (CONS (CAR EXPRES)
             (COND ((EQ (CAR EXPRES) (QUOTE LOG))
                (CONS (CADR EXPRES) (QUOTE ((B RAT8)))) )
               (T (QUOTE ((B RAT8)))) )))
           NIL ))
```

```
   (RATLOG EXP VAR (CONS (CONS (QUOTE A) EXPRES) ARG)) )
  (T
   (PROG (Y Z)
         (COND
          ((SETQ Y
            (FORM (COND ((EQ (CAR EXPRES) (QUOTE LOG)) (CADDR EXPRES))
              (T (CADR EXPRES)) )))
           (RETURN Y) )
          ((AND
            (EQ (CAR EXPRES) (QUOTE LOG))
            (SETQ Z (M2 (CADDR EXPRES) C NIL))
            (FREE (CADR EXPRES))
            (SETQ Y
             (M2
              EXP
              (QUOTE (TIMES (COEFFTT (C RAT8)) (COEFFTT (D ELEM))) )
              NIL )))
           (RETURN
            ((LAMBDA
              (A B C D BASE)
              (SUBST
               EXPRES
               VAR
               (INTEGRATE
                (SIMPTIMES (LIST
                  (SUBST
                   (LIST
                    (QUOTE QUOTIENT)
                    (LIST
                     (QUOTE DIFFERENCE)
                     (LIST (QUOTE EXPT) BASE VAR)
                     A )
                    B )
                   VAR
                   C )
                  (LIST
                   (QUOTE QUOTIENT)
                   (LIST (QUOTE EXPT) BASE VAR)
                   B )
                  (SUBST VAR EXPRES D) ))
                 VAR )))
             (CDR (SASSOC (QUOTE A) Z))
             (CDR (SASSOC (QUOTE B) Z))
             (CDR (SASSOC (QUOTE C) Y))
             (CDR (SASSOC (QUOTE D) Y))
             (CADR EXPRES) ))
           (T (RETURN NIL)) )))))
((OPTRIG (CAR EXPRES))
 (COND
  ((NOT (SETQ W (M2 (CADR EXPRES) C NIL))) (FORM (CADR EXPRES)))
  (T (PROG2 (SETQ POWERLIST T) (MONSTERTRIG EXP VAR (CADR EXPRES)))) ))
((FIXP1 (CADDR EXPRES)) (FORM (CADR EXPRES)))
((FREE (CADR EXPRES))
 (COND
  ((SETQ W
    (M2
     EXP
     (QUOTE (TIMES (COEFFTT (R RAT8)) (EXPT (BASE FREE) (P POLYP))) )
     NIL ))
   (CALLALGORT (SUBLIS W (QUOTE (R P BASE))) VAR) )
  ((M2 (CADDR EXPRES) C NIL) (SUPEREXPT EXP VAR (CADR EXPRES)))
```

```
      (T (FORM (CADDR EXPRES))) ))
    ((NOT (RAT8 (CADR EXPRES))) (FORM (CADR EXPRES)))
    ((AND (SETQ W (M2 (CADR EXPRES) RATROOTFORM NIL))
      (DENOMFIND (CADDR EXPRES)) )
     (PROG2 (SETQ POWERLIST T) (RATROOT EXP VAR (CADR EXPRES) W)) )
    ((NOT (INTEGERP1 (CADDR EXPRES)))
     (COND ((M2 EXP CHEBYFORM NIL) (CHEBY EXP VAR))
      (T (FORM (CADR EXPRES))) ))
    ((SETQ W (M2 (CADR EXPRES) D NIL))
     (COND
      ((SETQ ARG
        (M2
         EXP
         (QUOTE (TIMES
           (EXPT (VAR VARP) -1)
           (COEFFTT (AA FREE))
           (EXPT (SQ M1 D) -0.5E0) ))
         NIL ))
       (SIMP
        (SUBST
         (LIST (QUOTE EXPT) VAR -1)
         VAR
         (ALGEB2
          (LIST
           (QUOTE TIMES)
           -1
           (REPLACE ARG (QUOTE AA))
           (LIST
            (QUOTE EXPT)
            (SETQ Y
             (REPLACE ARG
              (QUOTE (PLUS (TIMES A (EXPT VAR 2)) (TIMES B VAR) C) )))
            -0.5E0 ))
          VAR
          Y
          (REPLACE ARG
           (QUOTE (((QUOTE* C) . A) ((QUOTE* B) . B) ((QUOTE* A) . C)) ))))))
      (T (ALGEB2 EXP VAR (CADR EXPRES) W)) ))
    ((SETQ W (M2 (CADR EXPRES) E NIL))
     (PROG2 (SETQ POWERLIST T) (ROOTLINPROD EXP VAR (CADR EXPRES) W)) )
    ((M2 EXP CHEBYFORM NIL) (CHEBY EXP VAR))
    ((NOT (M2 (SETQ W (EXPAND2 (CADR EXPRES))) (CADR EXPRES) NIL))
     (PROG2
      (SETQ EXP (SIMP (SUBST W (CADR EXPRES) EXP)))
      (FORM (SIMP (LIST (QUOTE EXPT) W (CADDR EXPRES)))) ))
    ((SETQ W (RATIONALIZE (CADR EXPRES)))
     (PROG2
      (SETQ EXP (SIMP (SUBST W (CADR EXPRES) EXP)))
      (FORM (SIMP (LIST (QUOTE EXPT) W (CADDR EXPRES)))) ))
    (T NIL) )))))

DEFINE
(((INTEGRATE
   (LAMBDA
    (EXP VAR)
    (PROG (Y ARG POWERLIST B W C D E RATROOTFORM CHEBYFORM)
          (COND ((FREE EXP) (RETURN (SIMPTIMES (LIST EXP VAR)))))
          (COND
           ((NOT (EQ (CAR EXP) (QUOTE PLUS))) (GO D))
           (T
            (RETURN
```

```
            (SIMPPLUS (MAPLIST (CDR EXP)
              (FUNCTION (LAMBDA (C) (INTEGRATE1 (CAR C))))) )))))
      D
            (COND ((SETQ Y (DIFFDIV EXP VAR)) (RETURN Y)))
            (SETQ Y
             (COND ((EQ (CAR EXP) (QUOTE TIMES)) (CDR EXP)) (T (LIST EXP))) )
            (SETQ C
             (QUOTE (PLUS (COEFFPT (B FREE) (X VARP)) (COEFFPT (A FREE)))) )
            (SETQ RATROOTFORM
             (QUOTE (TIMES
               (COEFFTT (E FREE))
               (PLUS (COEFFPT (A FREE) (VAR VARP)) (COEFFPT (B FREE)))
               (EXPT (PLUS (COEFFPT (C FREE) (VAR VARP)) (COEFFPT (D FREE)))
                -1 ))))
            (SETQ
             CHEBYFORM
             (QUOTE (TIMES
               (EXPT (VAR VARP) (R1 NUMBERP))
               (EXPT (PLUS (TIMES (COEFFTT (C2 FREE)) (EXPT (VAR VARP) (Q FREE1)))
                 (COEFFP (C1 FREE)) )
                (R2 NUMBERP) )
               (COEFFTT (A FREE)) )))
            (SETQ D
             (QUOTE (PLUS
               (COEFFPT (C FREE) (EXPT (X VARP) 2))
               (COEFFPT (B FREE) (X VARP))
               (COEFFPT (A FREE)) )))
            (SETQ E
             (QUOTE (TIMES (PLUS (COEFFPT (A FREE) (VAR VARP)) (COEFFPT (B FREE)))
               (PLUS (COEFFPT (C FREE) (VAR VARP)) (COEFFPT (D FREE))) )))
      LOOP
            (COND
             ((RAT8 (CAR Y)) (GO SKIP))
             ((SETQ W (FORM (CAR Y))) (RETURN W))
             (T (GO SPECIAL)) )
      SKIP
            (SETQ Y (CDR Y))
            (COND ((NULL Y)
              (RETURN (COND ((SETQ Y (POWERLIST EXP VAR)) Y)
                (T (MASTER (CONS VAR EXP))) ))))
            (GO LOOP)
      SPECIAL
            (RETURN (COND
              ((NOT (M2 EXP (SETQ Y (EXPAND2 EXP)) NIL)) (INTEGRATE Y VAR))
              ((AND (NOT POWERLIST) (SETQ Y (POWERLIST EXP VAR))) Y)
              ((SETQ Y (PARTS EXP VAR)) Y)
              (T (LIST (QUOTE INTEGRAL) EXP VAR)) )))))))

DEFINE
(((RAT8 (LAMBDA (EXP)
                (COND ((FREE EXP) T)
                      ((ATOM EXP) T)
                      ((MEMBER (CAR EXP) (QUOTE (PLUS TIMES)))
                       (AND (RAT8 (CADR EXP))
                            (COND ((CDDR EXP)
                                   (RAT8 (CONS (CAR EXP) (CDDR EXP))) )
                                  (T T) )))
                      ((NOT (EQ (CAR EXP) (QUOTE EXPT))) NIL)
                      ((FIXP1 (CADDR EXP)) (RAT8 (CADR EXP)))
                      (T NIL) )))))
```

```
DEFINE
(((INTEGRATE1 (LAMBDA (A) (INTEGRATE A VAR)))))

DEFINE
(((POLYP (LAMBDA (EXP)
    (COND
     ((FREE EXP) T)
     ((ATOM EXP) T)
     ((MEMBER (CAR EXP) (QUOTE (PLUS TIMES)))
      (AND (POLYP (CADR EXP))
       (OR (NULL (CDDR EXP)) (POLYP (CONS (CAR EXP) (CDDR EXP))))))
     ((EQ (CAR EXP) (QUOTE EXPT))
      (AND
       (NUMBERP (CADDR EXP))
       (INTEGERP (CADDR EXP))
       (GREATERP (CADDR EXP) 0)
       (POLYP (CADR EXP))))
     (T NIL) )))
  (CALLALGORT
   (LAMBDA
    (A VAR)
    (PROG NIL
          (FILEWRITE (QUOTE MANOVE) (QUOTE LISP) (QUOTE SUPERALGORT))
          (FILEAPND
           (QUOTE MANOVE)
           (QUOTE LISP)
           (LIST
            (TRANSL (CAR A))
            (TRANSL (SIMPTIMES (LIST (CADR A) (SIMPLOG (LIST (QUOTE E) (CADDR A)))))
            VAR ))
          (CHAIN (QUOTE ((SAVE MOSES T) (R FULMAN MANOVE))))
          (FILESEEK (QUOTE MANOVE) (QUOTE ANS))
          (RETURN (SIMP (UNTR (READ)))) )))))

DEFINE
(((SIN (LAMBDA (EXP VAR) (INTEGRATE (SIMP EXP) VAR)))
  (OPTRIG (LAMBDA (A) (MEMBER A (QUOTE (SIN COS SEC TAN CSC COT)))))
  (ELEM
   (LAMBDA
    (A)
    (COND
     ((FREE A) T)
     ((ATOM A) NIL)
     ((M2 A EXPRES NIL) T)
     (T (EVAL (CONS (QUOTE AND)
        (MAPLIST (CDR A) (FUNCTION (LAMBDA (C) (ELEM (CAR C))))) )
       NIL )))))))

DEFINE
(((FREE (LAMBDA (A)
                (COND ((ATOM A) (NOT (EQ A VAR)))
                      (T (AND (FREE (CAR A)) (FREE (CDR A)))) )))
  (VARP (LAMBDA (A) (EQ A VAR))) ))

DEFINE
(((DEFINITEINTEGRAL
   (LAMBDA (EXP VAR LOWER UPPER)
           (PROG (Y)
                 (SETQ Y (PRINT (INTEGRATE EXP VAR)))
                 (RETURN (SIMPDIFFERENCE (LIST (SUBST UPPER VAR Y)
                                               (SUBST LOWER VAR Y) ))))))
```

```
  (DOUBLEINTEGRAL
   (LAMBDA (EXP L)
           (PROG (Y)
                 (SETQ Y
                       (DEFINITEINTEGRAL EXP
                                         (CAAR L)
                                         (CADAR L)
                                         (CAR (CDDAR L)) ))
                 (RETURN (DEFINITEINTEGRAL Y
                                           (CAADR L)
                                           (CADADR L)
                                           (CAR (CDDADR L)) )))))))

DEFINE
(((INTEGRALLOOKUP
   (LAMBDA
    (EXP)
    (COND
     ((EQ (CAR EXP) (QUOTE LOG))
      (SIMP (SUBST
        (CADDR EXP)
        (QUOTE X)
        (QUOTE (PLUS (TIMES X (LOG E X)) (TIMES -1 X))) )))
     ((EQ (CAR EXP) (QUOTE PLUS)) (SIMPTIMES (LIST  0.5E0 EXP EXP)))
     ((EQ (CAR EXP) (QUOTE EXPT))
      (COND
       ((FREE (CADR EXP))
        (SIMPTIMES (SUBST
          EXP
          (QUOTE A)
          (SUBST (CADR EXP) (QUOTE B) (QUOTE (A (EXPT (LOG E B) -1)))) )))
       ((EQP (CADDR EXP) -1)
        (SIMP (SUBST (CADR EXP) (QUOTE X) (QUOTE (LOG E X)))) )
       (T (SIMP (SUBST
          (SIMPPLUS (LIST (CADDR EXP) 1))
          (QUOTE N)
          (SUBST
           (CADR EXP)
           (QUOTE X)
           (QUOTE (TIMES (EXPT N -1) (EXPT X N))) ))))))
      (T (SUBST
        (CADR EXP)
        (QUOTE X)
        (COR (SASSOC
          (CAR EXP)
          (QUOTE ((SIN TIMES -1 (COS X))
            (COS SIN X)
            (TAN LOG E (SEC X))
            (SEC LOG E (PLUS (SEC X) (TAN X)))
            (COT LOG E (SIN X))
            (CSC LOG E (PLUS (SEC X) (TAN X))) ))
          (QUOTE NILL) )))))))
  (DIFFDIV
   (LAMBDA
    (EXP VAR)
    (PROG (Y A X V D Z W R)
          (SETQ X
           (M2
            EXP
            (QUOTE (TIMES (COEFFTT (A FREE)) (COEFFTT (B TRUE))))
            NIL ))
```

```
      (SETQ A (CDR (SASSOC (QUOTE A) X)))
      (SETQ EXP (CDR (SASSOC (QUOTE B) X)))
      (COND
       ((AND
          (EQ (CAR EXP) (QUOTE EXPT))
          (EQ (CAADR EXP) (QUOTE PLUS))
          (INTEGERP (CADDR EXP))
          (LESSP (CADDR EXP) 6)
          (GREATERP (CADDR EXP) 0) )
        (RETURN (SIMPTIMES (LIST A
           (INTEGRATE (EXPANDEXPT (CADR EXP) (CADDR EXP)) VAR) )))))
      (SETQ EXP
       (COND ((EQ (CAR EXP) (QUOTE TIMES)) EXP)
        (T (LIST (QUOTE TIMES) EXP)) ))
      (SETQ Z (CDR EXP))
A
      (SETQ Y (CAR Z))
      (SETQ R
       (LIST (QUOTE PLUS)
        (CONS (QUOTE COEFFPT)
         (CONS (QUOTE (C FREE1)) (CHOICE Y (CDR EXP))) )))
      (COND
       ((SETQ W (M2 (DIFF1 Y VAR) R NIL))
        (RETURN
         (SIMPTIMES
          (LIST
           Y
           A
           Y
           (SIMPEXPT (LIST (SIMPTIMES (LIST 2 (CDR (SASSOC (QUOTE C) W))))
             -1 ))))))
       (COND
        ((MEMBER (CAR Y) (QUOTE (EXPT LOG)))
         (COND
          ((FREE (CADR Y)) (SETQ W (CADDR Y)))
          ((FREE (CADDR Y)) (SETQ W (CADR Y)))
          (T (SETQ W 0)) ))
        ((MEMBER (CAR Y) (QUOTE (PLUS TIMES))) (SETQ W Y))
        (T (SETQ W (CADR Y))) )
       (COND
        ((SETQ
          W
          (COND
           ((AND
             (EQ (CAR (SETQ X (DIFF1 W VAR))) (QUOTE PLUS))
             (EQ
              (CAR (SETQ V (CAR (SETQ D (CHOICE Y (CDR EXP))))))
              (QUOTE PLUS) )
             (NOT (CDR D)) )
            (COND ((SETQ D (MATCHSUM (CDR X) (CDR V)))
              (LIST (CONS (QUOTE C) D)) )
             (T NIL) ))
           (T (M2 X R NIL)) ))
         (RETURN
          (COND
           ((NULL (SETQ X (INTEGRALLOOKUP Y))) NIL)
           (T
            (SIMPTIMES
             (LIST
              X
              A
```

```
                (COND
                 ((EQ W T) 1)
                 (T (SIMPEXPT (LIST (CDR (SASSOC (QUOTE C) W)) -1))) ))))))))
          (SETQ Z (CDR Z))
          (COND ((NULL Z) (RETURN NIL)))
          (GO A) )))))

DEFINE
(((TRUE (LAMBDA (A) T))))

DEFINE
(((MATCHSUM
   (LAMBDA
    (ALIST BLIST)
    (PROG (R S C D)
          (SETQ S
           (M2
            (CAR ALIST)
            (QUOTE (TIMES (COEFFTT (A FREE)) (COEFFTT (C TRUE))))
            NIL ))
          (SETQ C (CDR (SASSOC (QUOTE C) S)))
          (COND
           ((NOT (SETQ R
                  (M2
                   (CONS (QUOTE PLUS) BLIST)
                   (LIST
                    (QUOTE PLUS)
                    (CONS (QUOTE TIMES)
                     (CONS
                      (QUOTE (COEFFTT (B FREE1)))
                      (COND ((EQ (CAR C) (QUOTE TIMES)) (CDR C))
                       (T (LIST C)) )))
                    (QUOTE (D TRUE)) )
                   NIL )))
            (RETURN NIL) ))
          (SETQ D
           (SIMP (LIST
             (QUOTE TIMES)
             (SUBLIS S (QUOTE A))
             (LIST (QUOTE EXPT) (SUBLIS R (QUOTE B)) -1) )))
          (COND ((M2 (CONS (QUOTE PLUS) ALIST) (TIMESLOOP D BLIST) NIL)
            (RETURN D) )
           (T (RETURN NIL)) ))))))

DEFINE
(((EXPANDEXPT (LAMBDA (A N)
                      (PROG (Y)
                            (SETQ Y A)
                       LOOP
                            (SETQ N (SUB1 N))
                            (COND ((ZEROP N) (RETURN Y)))
                            (SETQ Y
                                  (EXPAND (CDR A)
                                          (COND ((EQ (CAR Y)
                                                     (QUOTE PLUS) )
                                                 (CDR Y) )
                                                (T (LIST Y)) )))
                            (GO LOOP) )))))
```

METHODS 1-9 OF SIN'S SECOND STAGE

```
DEFINE
(((SUPEREXPT
   (LAMBDA
    (EXP VAR BASE)
    (PROG (EXPTFLAG Y W)
          (SETQ Y (ELEMXPT EXP))
          (COND (EXPTFLAG (RETURN NIL)))
          (RETURN
           (SIMP
            (SUBST
             (LIST (QUOTE EXPT) BASE VAR)
             VAR
             (INTEGRATE
              (SIMPQUOTIENT
               (LIST Y
                (SIMPTIMES (LIST VAR (SIMPLOG (LIST (QUOTE E) BASE)))) ))
              VAR ))))))))
  (ELEMXPT
   (LAMBDA
    (EXP)
    (COND
     ((FREE EXP) EXP)
     ((ATOM EXP) (SETQ EXPTFLAG T))
     ((NOT (EQ (CAR EXP) (QUOTE EXPT)))
      (CONS (CAR EXP)
       (MAPLIST (CDR EXP) (FUNCTION (LAMBDA (C) (ELEMXPT (CAR C))))) ))
     ((NOT (FREE (CADR EXP)))
      (LIST (QUOTE EXPT) (ELEMXPT (CADR EXP)) (ELEMXPT (CADDR EXP))) )
     ((NOT (EQ (CADR EXP) BASE))
      (ELEMXPT (LIST
         (QUOTE EXPT)
         BASE
         (SIMP (LIST
           (QUOTE TIMES)
           (LIST (QUOTE LOG) BASE (CADR EXP))
           (CADDR EXP) )))))
     ((NOT (SETQ W
         (M2
          (CADDR EXP)
          (QUOTE (PLUS (COEFFPT (A FREE) (VAR VARP)) (COEFFPT (B FREE))))
          NIL )))
      (LIST (CAR EXP) BASE (ELEMXPT (CADDR EXP))) )
     (T (SIMP (SUBST
         BASE
         (QUOTE BASE)
         (SUBLIS W (QUOTE (TIMES (EXPT BASE B) (EXPT VAR A)))) ))))))))

DEFINE
(((SUBST10
   (LAMBDA (EXP)
    (COND
     ((ATOM EXP) EXP)
     ((AND (EQ (CAR EXP) (QUOTE EXPT)) (EQ (CADR EXP) VAR))
      (LIST (CAR EXP) VAR (INTEGERP (QUOTIENT (CADDR EXP) D))) )
     (T (MAPLIST EXP (FUNCTION (LAMBDA (C) (SUBST10 (CAR C))))))) )))
  (POWERLIST
   (LAMBDA
    (EXP VAR)
```

```
    (PROG (Y Z C D POWERLIST B)
          (SETQ Y
           (M2
            EXP
            (QUOTE (TIMES
              (EXPT (VAR VARP) (C INTEGERP))
              (COEFFTT (A FREE))
              (COEFFTT (B TRUE)) ))
            NIL ))
          (SETQ B (CDR (SASSOC (QUOTE B) Y)))
          (SETQ C (CDR (SASSOC (QUOTE C) Y)))
          (COND ((NOT (SETQ Z (RAT1O B))) (RETURN NIL)))
          (SETQ D (LISTGCD (CONS (ADD1 C) POWERLIST)))
          (COND ((NULL D) (RETURN NIL)))
          (RETURN
           (SIMP
            (SUBST
             (LIST (QUOTE EXPT) VAR D)
             VAR
             (INTEGRATE5 (SIMP (LIST
                (QUOTE TIMES)
                (EXPT D -1)
                (CDR (SASSOC (QUOTE A) Y))
                (LIST (QUOTE EXPT) VAR (SUB1 (QUOTIENT (ADD1 C) D)))
                (SUBST1O B) ))
              VAR ))))))
  (RAT1O (LAMBDA (EXP)
    (COND
     ((FREE EXP) T)
     ((ATOM EXP) NIL)
     ((EQ (CAR EXP) (QUOTE EXPT))
      (COND
       ((EQ (CADR EXP) VAR)
        (COND ((INTEGERP (CADDR EXP))
          (SETQ POWERLIST (CONS (CADDR EXP) POWERLIST)) )
         (T NIL) ))
       (T (AND (RAT1O (CADR EXP)) (RAT1O (CADDR EXP))))) ))
     ((MEMBER (CAR EXP) (QUOTE (PLUS TIMES)))
      (AND (RAT1O (CADR EXP))
       (OR (NULL (CDDR EXP)) (RAT1O (CONS (CAR EXP) (CDDR EXP))))) ))
     ((EQ (CAR EXP) (QUOTE LOG)) (RAT1O (CADDR EXP)))
     (T (RAT1O (CADR EXP))) )))
  (LISTGCD (LAMBDA (POWERLIST)
    (PROG (D)
          (SETQ D (CAR POWERLIST))
     LOOP
          (SETQ POWERLIST (CDR POWERLIST))
          (COND ((ONEP D) (RETURN NIL)))
          (COND ((NULL POWERLIST) (RETURN D)))
          (SETQ D (GCD D (CAR POWERLIST)))
          (GO LOOP) ))))

DEFINE
(((INTEGRATE5 (LAMBDA (EXP VAR)
                      (COND ((RAT8 EXP) (MASTER (CONS VAR EXP)))
                            (T (INTEGRATE EXP VAR)) )))))

DEFINE
(((ABSOLUTE (LAMBDA (A) (COND ((LESSP A 0) (MINUS A)) (T A))))))
```

```
DEFINE
(((INTEGERP (LAMBDA (A)
    (PROG (Y)
          (SETQ Y 1)
          (COND ((NOT (NUMBERP A)) (RETURN NIL))
           ((NOT (FLOATP A)) (RETURN A)) )
     C
          (COND
           ((EQP Y A) (RETURN Y))
           ((LESSP Y A) (GO A))
           ((NOT (GREATERP (DIFFERENCE Y A)  0.98999999E0)) (RETURN NIL)) )
          (SETQ Y (SUB1 Y))
          (GO C)
     A
          (COND ((NOT (GREATERP (DIFFERENCE A Y)  0.98999999E0)) (RETURN NIL)) )
          (SETQ Y (ADD1 Y))
          (GO C) )))
  (FIXP1 (LAMBDA (A) (AND (NUMBERP A) (FIXP A))))
  (RAT3 (LAMBDA (EXP IND)
    (COND
     ((FREE EXP) T)
     ((ATOM EXP) IND)
     ((MEMBER (CAR EXP) (QUOTE (TIMES PLUS)))
      (AND (RAT3 (CADR EXP) IND)
       (OR (NULL (CDDR EXP)) (RAT3 (CONS (CAR EXP) (CDDR EXP)) IND)) ))
     ((NOT (EQ (CAR EXP) (QUOTE EXPT)))
      (COND ((EQ (CAR EXP) (QUOTE LOG)) (RAT3 (CDDR EXP) T))
       (T (RAT3 (CADR EXP) T)) ))
     ((FREE (CADR EXP)) (RAT3 (CADDR EXP) T))
     ((FIXP1 (CADDR EXP)) (RAT3 (CADR EXP) IND))
     ((AND (M2 (CADR EXP) RATROOT NIL) (DENOMFIND (CADDR EXP)))
      (SETQ ROOTLIST (CONS (DENOMFIND (CADDR EXP)) ROOTLIST)) )
     (T (RAT3 (CADR EXP) NIL)) )))
  (SUBST4 (LAMBDA (EXP)
    (COND
     ((FREE EXP) EXP)
     ((ATOM EXP) A)
     ((NOT (EQ (CAR EXP) (QUOTE EXPT)))
      (MAPLIST EXP (FUNCTION (LAMBDA (C) (SUBST4 (CAR C))))) )
     ((M2 (CADR EXP) RATROOT NIL)
      (LIST (CAR EXP) B (INTEGERP (TIMES K (CADDR EXP)))) )
     (T (LIST (CAR EXP) (SUBST4 (CADR EXP)) (SUBST4 (CADDR EXP)))) )))
  (FINDINGK (LAMBDA (LIST)
    (PROG (K)
          (SETQ K 1)
     A
          (COND ((NULL LIST) (RETURN K)))
          (SETQ K (QUOTIENT (TIMES K (CAR LIST)) (GCD K (CAR LIST))))
          (SETQ LIST (CDR LIST))
          (GO A) )))
  (DENOMFIND (LAMBDA (K)
    (PROG (Y)
          (COND ((NOT (NUMBERP K)) (RETURN NIL)))
          (SETQ Y 1)
     A
          (COND ((INTEGERP (TIMES K Y)) (RETURN Y)))
          (SETQ Y (ADD1 Y))
          (COND ((LESSP Y 25) (GO A)))
          (RETURN NIL) )))
  (GCD (LAMBDA (A B)
    (PROG NIL
```

```
    A
          (COND ((ZEROP A) (RETURN (ABSOLUTE B))))
          (SETQ B (REMAINDER B A))
          (COND ((ZEROP B) (RETURN (ABSOLUTE A))))
          (SETQ A (REMAINDER A B))
          (GO A) )))
  (RATROOT
   (LAMBDA
    (EXP VAR RATROOT W)
    (PROG (ROOTLIST K Y W1)
          (COND ((SETQ Y (CHEBY EXP VAR)) (RETURN Y)))
          (COND ((NOT (RAT3 EXP T)) (RETURN NIL)))
          (SETQ K (FINDINGK ROOTLIST))
          (SETQ W1 (CONS (CONS (QUOTE K) K) W))
          (SETQ
           Y
           (SUBST41
            EXP
            (SIMP (SUBLIS W1
              (QUOTE (QUOTIENT
                (DIFFERENCE B (TIMES D (EXPT VAR K)))
                (DIFFERENCE (TIMES C (EXPT VAR K)) A) ))))
            VAR ))
          (SETQ
           Y
           (INTEGRATE
            (SIMP
             (LIST
              (QUOTE TIMES)
              Y
              (SUBLIS
               W1
               (QUOTE (QUOTIENT
                 (TIMES E
                  (DIFFERENCE
                   (TIMES A D K (EXPT VAR (PLUS -1 K)))
                   (TIMES B C K (EXPT VAR (PLUS -1 K))) ))
                 (EXPT (DIFFERENCE (TIMES C (EXPT VAR K)) A) 2) )))))
            VAR ))
          (RETURN (SIMP (SUBST
             (SIMP (LIST (QUOTE EXPT) RATROOT (LIST (QUOTE EXPT) K -1)))
             VAR
             Y ))))))))

DEFINE
(((SUBST41 (LAMBDA (EXP A B) (SUBST4 EXP)))))

DEFINE
(((CHEBY
   (LAMBDA
    (EXP VAR)
    (PROG (R1 R2 D1 D2 N1 N2 W Q)
          (COND
           ((NOT
             (SETQ
              W
              (M2
               EXP
               (QUOTE (TIMES
                 (EXPT (VAR VARP) (R1 NUMBERP))
                 (EXPT (PLUS (TIMES (COEFFTT (C2 FREE))
```

```
          (EXPT (VAR VARP) (Q FREE1)) )
         (COEFFP (C1 FREE)) )
        (R2 NUMBERP) )
       (COEFFTT (A FREE)) ))
     NIL )))
  (RETURN NIL) ))
(SETQ Q (CDR (SASSOC (QUOTE Q) W)))
(SETQ
 W
 (CONS
  (CONS (QUOTE A)
   (SIMPQUOTIENT (LIST (CDR (SASSOC (QUOTE A) W)) Q)) )
  (CONS
   (CONS
    (QUOTE R1)
    (SIMPQUOTIENT (LIST (SIMPPLUS (LIST
        1
        (SIMPMINUS (LIST Q))
        (CDR (SASSOC (QUOTE R1) W)) ))
      Q )))
   W )))
(SETQ R1 (CDR (SASSOC (QUOTE R1) W)))
(SETQ R2 (CDR (SASSOC (QUOTE R2) W)))
(SETQ W (REVERSE W))
(COND
 ((NOT (AND
    (SETQ D1 (DENOMFIND R1))
    (SETQ D2 (DENOMFIND R2))
    (SETQ N1 (INTEGERP (TIMES R1 D1)))
    (SETQ N2 (INTEGERP (TIMES R2 D2)))
    (SETQ W
     (CONS (CONS (QUOTE D1) D1)
      (CONS (CONS (QUOTE D2) D2)
       (CONS (CONS (QUOTE N1) N1)
        (CONS (CONS (QUOTE N2) N2) W) ))))))
  (RETURN NIL) )
 ((AND (INTEGERP R1) (GREATERP R1 0))
  (RETURN
   (SIMP
    (SUBST
     (SUBLIS W (QUOTE (PLUS C1 (TIMES C2 (EXPT VAR Q)))))
     VAR
     (INTEGRATE
      (EXPAND
       (SUBLIS W
        (QUOTE ((TIMES
           A
           (EXPT VAR R2)
           (EXPT C2 (MINUS (PLUS R1 1))) ))))
       (CDR (EXPANDEXPT (SUBLIS W (QUOTE (PLUS VAR (TIMES -1 C1))))
         R1 )))
       VAR ))))))
 ((INTEGERP R2)
  (RETURN
   (SIMP
    (SUBST
     (SUBLIS W (QUOTE (EXPT VAR (QUOTIENT Q D1))))
     VAR
     (MASTER
      (CONS
       VAR
```

```
                (SIMP
                 (SUBLIS W
                  (QUOTE (TIMES
                    D1
                    A
                    (EXPT VAR (PLUS N1 D1 -1))
                    (EXPT (PLUS (TIMES C2 (EXPT VAR D1)) C1) R2) ))))))))))
          ((AND (INTEGERP R1) (LESSP R1 0))
            (RETURN
            (SIMP
             (SUBST
              (SUBLIS W
               (QUOTE (EXPT (PLUS C1 (TIMES C2 (EXPT VAR Q)))
                 (QUOTIENT 1 D2) )))
              VAR
              (MASTER
               (CONS
                VAR
                (SIMP (SUBLIS W
                  (QUOTE (TIMES
                    A
                    D1
                    (EXPT C2 (MINUS (PLUS R1 1)))
                    (EXPT VAR (PLUS N1 D1 -1))
                    (EXPT (DIFFERENCE (EXPT VAR D1) C1) R1) ))))))))))
          ((INTEGERP (SIMPPLUS (LIST R1 R2)))
           (RETURN
            (SIMP
             (SUBST
              (SUBLIS W
               (QUOTE (EXPT (QUOTIENT (PLUS C1 (TIMES C2 (EXPT VAR Q)))
                  (EXPT VAR Q) )
                 (QUOTIENT 1 D1) )))
              VAR
              (MASTER
               (CONS
                VAR
                (SIMP (SUBLIS W
                  (QUOTE (TIMES
                    -1
                    A
                    D1
                    (EXPT C1 (PLUS R1 R2 1))
                    (EXPT VAR (PLUS N2 D1 -1))
                    (EXPT (DIFFERENCE (EXPT VAR D1) C2)
                     (TIMES -1 (PLUS R1 R2 2)) ))))))))))))
          (T (RETURN NIL)) ))))))

DEFINE
(((ALGEB (LAMBDA (A B C D) (ALGEB2 A B C (CONS NIL D))))))

DEFINE
(((ALGEB2
   (LAMBDA
    (EXP VAR SQUARE W)
    (PROG (A Y B C F1 A1 Y1 X1 E D H G)
          (SETQ A (CDR (SASSOC (QUOTE A) W)))
          (SETQ B (CDR (SASSOC (QUOTE B) W)))
          (SETQ C (CDR (SASSOC (QUOTE C) W)))
          (COND ((NOT (RAT6 EXP)) (RETURN NIL)))
          (SETQ Y1
```

```
      (SIMP (LIST
        (QUOTE PLUS)
        VAR
        (LIST (QUOTE QUOTIENT) B (LIST (QUOTE TIMES) 2 C)) )))
     (SETQ X1
      (SIMP (LIST
        (QUOTE DIFFERENCE)
        VAR
        (LIST (QUOTE QUOTIENT) B (LIST (QUOTE TIMES) 2 C)) )))
     (SETQ A1
      (SIMP (LIST
        (QUOTE DIFFERENCE)
        A
        (LIST
         (QUOTE QUOTIENT)
         (LIST (QUOTE EXPT) B 2)
         (LIST (QUOTE TIMES) 4 C) ))))
     (COND
      ((AND (NUMBERP C) (GREATERP C 0)) (GO L1))
      ((AND (NUMBERP C) (LESSP C 0)) (GO L2))
      ((ASKPOS C) (GO L1))
      ((ASKNEG C) (GO L2))
      ((ASKIT C (QUOTE POSITIVE)) (GO L1))
      ((ASKIT C (QUOTE NEGATIVE)) (GO L2))
      (T (RETURN (ALGEB EXP VAR SQUARE W))) )
L1
     (COND
      ((AND (NUMBERP A1) (GREATERP A1 0)) (GO L3))
      ((AND (NUMBERP A1) (LESSP A1 0)) (GO L5))
      ((AND (NUMBERP A1) (ZEROP A1)) (GO L4))
      ((ASKPOS A1) (GO L3))
      ((ASKNEG A1) (GO L5))
      ((ASKIT A1 (QUOTE POSITIVE)) (GO L3))
      ((ASKIT A1 (QUOTE NEGATIVE)) (GO L5))
      ((ASKZERO A1) (GO L4))
      (T (RETURN (ALGEB EXP VAR SQUARE W))) )
L2
     (COND
      ((AND (NUMBERP A1) (GREATERP A1 0)) (GO L6))
      ((AND (NUMBERP A1) (LESSP A1 0))
       (RETURN (ALGEB EXP VAR SQUARE W)) )
      ((ASKPOS A1) (GO L6))
      ((ASKIT A1 (QUOTE POSITIVE)) (GO L6))
      (T (RETURN (ALGEB EXP VAR SQUARE W))) )
L4
     (SETQ C (SIMPEXPT (LIST C  0.5E0)))
     (SETQ Y (SUBST6 EXP X1 (SIMP (LIST (QUOTE TIMES) C VAR))))
     (SETQ Y (INTEGRATE (SIMP Y) VAR))
     (RETURN (SIMP (SUBST Y1 VAR Y)))
L3
     (SETQ H (QUOTE (ARCTAN X)))
     (SETQ E (QUOTE (TAN X)))
     (SETQ F1 (QUOTE (SEC X)))
     (SETQ G (QUOTE (EXPT (SEC X) 2)))
     (GO GETOUT)
L5
     (SETQ H (QUOTE (ARCSEC X)))
     (SETQ E (QUOTE (SEC X)))
     (SETQ A1 (SIMPMINUS (LIST A1)))
     (SETQ F1 (QUOTE (TAN X)))
     (SETQ G (QUOTE (TIMES (TAN X) (SEC X))))
```

```
          (GO GETOUT)
     L6
          (SETQ E (QUOTE (SIN X)))
          (SETQ G (QUOTE (COS X)))
          (SETQ C (SIMPMINUS (LIST C)))
          (SETQ H (QUOTE (ARCSIN X)))
          (SETQ F1 (QUOTE (COS X)))
     GETOUT
          (SETQ C (SIMPEXPT (LIST (SIMPQUOTIENT (LIST A1 C))  0.5E0)))
          (SETQ D (SIMPEXPT (LIST A1  0.5E0)))
          (SETQ
           Y
           (SUBST6
            EXP
            (EXPAND2 (SIMP (LIST (QUOTE PLUS)
                                 (LIST (QUOTE TIMES)
                                       C
                                       (SUBST VAR (QUOTE X) E) )
                                 (SIMPDIFFERENCE (LIST X1 VAR)) )))
            (SIMP (LIST (QUOTE TIMES) D (SUBST VAR (QUOTE X) F1))) ))
          (SETQ Y (SIMP (LIST (QUOTE TIMES) C (SUBST VAR (QUOTE X) G) Y)))
          (SETQ Y (INTEGRATE (EXPAND2 Y) VAR))
          (RETURN (SIMP (SUBST
             (SUBST
              (LIST (QUOTE TIMES) (LIST (QUOTE EXPT) C -1) Y1)
              (QUOTE X)
              H )
             VAR
             Y ))))))))

DEFINE
(((ASKIT (LAMBDA (A B)
    (AND
     (NOT (PRIN1 (QUOTE IS)))
     (NOT (PRIN1 BLANK))
     (PRINT A)
     (PRINT B)
     (EQ (RDFLX) (QUOTE YES)) )))
  (ASKZERO (LAMBDA (A) (ASKIT A (QUOTE ZERO))))
  (ASKPOS
   (LAMBDA
    (A)
    (M2
     A
     (QUOTE (PLUS (COEFFPT
        (B (FUNCTION (LAMBDA (B) (AND (NUMBERP B) (GREATERP B 0)))))
        (COEFFTT (C POSFN)) )))
     NIL )))
  (ASKNEG
   (LAMBDA
    (A)
    (M2
     A
     (QUOTE (PLUS (COEFFPT
        (B (FUNCTION (LAMBDA (B) (AND (NUMBERP B) (LESSP B 0)))))
        (COEFFTT (C POSFN)) )))
     NIL )))
  (POSFN (LAMBDA (C)
    (COND
     ((ATOM C) (GET C (QUOTE POSITIVE)))
     ((EQ (CAR C) (QUOTE EXPT))
```

```
      (COND
       ((NUMBERP (CADDR C)) (INTEGERP (QUOTIENT (CADDR C) 2)))
       ((ATOM (CADR C)) (GET (CADR C) (QUOTE POSITIVE)))
       (T NIL) ))
     (T NIL) )))))

DEFINE
(((PFCTSQ (LAMBDA (X)
    (PROG (Y)
          (SETQ Y 1)
     A
          (COND ((EQP (TIMES Y Y) X) (RETURN Y))
           ((GREATERP (TIMES Y Y) X) (RETURN NIL)) )
          (SETQ Y (ADD1 Y))
          (GO A) )))
  (RAT6 (LAMBDA (EXP)
    (COND
     ((FREE EXP) T)
     ((ATOM EXP) T)
     ((MEMBER (CAR EXP) (QUOTE (PLUS TIMES)))
      (AND (RAT6 (CADR EXP))
       (OR (NULL (CDDR EXP)) (RAT6 (CONS (CAR EXP) (CDDR EXP)))) ))
     ((NOT (EQ (CAR EXP) (QUOTE EXPT))) NIL)
     ((FIXP1 (CADDR EXP)) (RAT6 (CADR EXP)))
     ((NOT (INTEGERP (SIMPTIMES (LIST 2 (CADDR EXP))))) NIL)
     (T (M2 (CADR EXP) SQUARE NIL)) )))
  (SUBST6
   (LAMBDA
    (EXP A B)
    (COND ((FREE EXP) EXP)
          ((ATOM EXP) A)
          ((MEMBER (CAR EXP) (QUOTE (PLUS TIMES)))
           (CONS (CAR EXP)
                 (MAPLIST (CDR EXP)
                          (FUNCTION (LAMBDA (C) (SUBST6 (CAR C) A B))) )))
          ((NOT (EQ (CAR EXP) (QUOTE EXPT))) (ERROR))
          ((FIXP1 (CADDR EXP))
           (LIST (CAR EXP) (SUBST6 (CADR EXP) A B) (CADDR EXP)) )
          (T (LIST (CAR EXP) B (INTEGERP (TIMES 2 (CADDR EXP))))) )))
  (TRIGSQRT
   (LAMBDA
    (EXP VAR SQUARE W)
    (PROG (Y A B C D E F1 G H)
          (SETQ A (CDR (SASSOC (QUOTE A) W)))
          (SETQ B (CDR (SASSOC (QUOTE B) W)))
          (COND ((OR (NOT (NUMBERP A)) (NOT (NUMBERP B)))
                 (RETURN (ALGEB EXP VAR SQUARE W)) )
                ((NOT (RAT6 EXP)) (RETURN NIL)) )
          (COND ((GREATERP A 0)
                 (COND ((GREATERP B 0)
                        (AND (SETQ H (QUOTE (ARCTAN X)))
                             (SETQ E (QUOTE (TAN X)))
                             (SETQ F1 (QUOTE (SEC X)))
                             (SETQ G (QUOTE (EXPT (SEC X) 2))) ))
                       (T (AND (SETQ E (QUOTE (SIN X)))
                               (SETQ G (QUOTE (COS X)))
                               (SETQ B (MINUS B))
                               (SETQ F1 (QUOTE (COS X)))
                               (SETQ H (QUOTE (ARCSIN X))) ))))
                (T (AND (SETQ E (QUOTE (SEC X)))
                        (SETQ A (MINUS A))
```

```
                    (SETQ F1 (QUOTE (TAN X)))
                    (SETQ G (QUOTE (TIMES (TAN X) (SEC X))))
                    (SETQ H (QUOTE (ARCSEC X))) )))
          (COND ((NOT (SETQ C (PFCTSQ (QUOTIENT A B)))) (RETURN NIL))
                ((NOT (SETQ D (PFCTSQ A))) (RETURN NIL)) )
          (SETQ Y
                (SUBST6 EXP
                        (SIMP (LIST (QUOTE TIMES)
                                    C
                                    (SUBST VAR (QUOTE X) E) ))
                        (SIMP (LIST (QUOTE TIMES)
                                    D
                                    (SUBST VAR (QUOTE X) F1) ))))
          (SETQ Y (SIMP (LIST (QUOTE TIMES) C (SUBST VAR (QUOTE X) G) Y)))
          (SETQ Y (TRIGINT Y VAR))
          (RETURN (SIMP (SUBST (SUBST (LIST (QUOTE TIMES)
                                            (LIST (QUOTE EXPT) C -1)
                                            VAR )
                                      (QUOTE X)
                                      H )
                               VAR
                               Y ))))))))

DEFINE
(((ALGEB
   (LAMBDA
    (EXP VAR SQUARE W)
    (PROG (A B C A1 C1 Y PROBL)
          (SETQ A (CDR (SASSOC (QUOTE A) W)))
          (SETQ B (CDR (SASSOC (QUOTE B) W)))
          (SETQ C (CDR (SASSOC (QUOTE C) W)))
          (COND ((NOT (RAT6 EXP)) (RETURN NIL)))
          (COND
           ((AND (NOT (NUMBERP C)) (ASK C))
            (SETQ C1 (SIMPEXPT (LIST C  0.5E0))) )
           ((NOT (NUMBERP C)) (GO A))
           ((NOT (GREATERP C 0)) (GO A))
           (T (SETQ C1 (SIMPSQRT C))) )
          (SETQ Y
           (SUBST6
            EXP
            (SUBSTL (A B C1 VAR)
             (QUOTIENT (DIFFERENCE (EXPT VAR 2) A)
              (PLUS B (TIMES 2 (TIMES VAR C1))) ))
            (SUBSTL (A B VAR C1)
             (QUOTIENT (PLUS (TIMES (EXPT VAR 2) C1) (TIMES B VAR) (TIMES A C1))
              (PLUS B (TIMES 2 (TIMES VAR C1))) ))))
          (SETQ
           PROBL
           (LIST
            (QUOTE TIMES)
            Y
            (SUBSTL (A B C1 VAR)
             (TIMES 2
              (TIMES (PLUS (TIMES B VAR) (TIMES (EXPT VAR 2) C1) (TIMES A C1))
               (EXPT (PLUS B (TIMES 2 (TIMES VAR C1))) -2) )))))
          (SETQ Y
           (SUBSTL (VAR C1 SQUARE)
            (PLUS (TIMES VAR C1) (EXPT SQUARE (QUOTIENT 1 2))) ))
          (GO B)
      A
```

```
          (COND
           ((AND (NOT (NUMBERP A)) (ASK A))
            (SETQ A1 (SIMPEXPT (LIST A  0.5E0))) )
           ((NOT (NUMBERP A)) (ERROR (QUOTE (NOT YET))))
           ((LESSP A 0) (ERROR (QUOTE (NOT YET))))
           (T (SETQ A1 (SIMPSQRT A))) )
          (SETQ Y
           (SUBST6
            EXP
            (SUBSTL (B C A1 VAR)
             (QUOTIENT (DIFFERENCE (TIMES 2 (TIMES VAR A1)) B)
              (DIFFERENCE C (EXPT VAR 2)) ))
            (SUBSTL (B C A1 VAR)
             (QUOTIENT (PLUS
               (TIMES A1 (EXPT VAR 2))
               (TIMES -1 (TIMES B VAR))
               (TIMES A1 C) )
              (DIFFERENCE C (EXPT VAR 2)) ))))
          (SETQ
           PROBL
           (LIST
            (QUOTE TIMES)
            Y
            (SUBSTL (B C A1 VAR)
             (TIMES
              (TIMES 2
               (PLUS
                (TIMES A1 (EXPT VAR 2))
                (TIMES -1 (TIMES B VAR))
                (TIMES A1 C) ))
              (EXPT (DIFFERENCE C (EXPT VAR 2)) -2) ))))
          (SETQ Y
           (SUBSTL (VAR A1 SQUARE)
            (QUOTIENT (DIFFERENCE (EXPT SQUARE (QUOTIENT 1 2)) A1) VAR) ))
     B
          (RETURN (SIMP (UNTR (SUBST Y VAR (MASTER (CONS VAR PROBL))))) )))))

DEFLIST
(((SUBSTL
   (LAMBDA (A ALIST)
           (SUBLIS (MAPLIST (CAR A)
                            (FUNCTION (LAMBDA (B)
                                              (CONS (CAR B)
                                                    (EVAL (CAR B) ALIST) ))))
                   (CADR A) ))))
 FEXPR )

DEFINE
(((SIMPSQRT (LAMBDA (X)
                    (PROG (Y)
                          (SETQ Y 1)
                     A
                          (COND ((EQP (TIMES Y Y) X) (RETURN Y))
                                ((GREATERP (TIMES Y Y) X)
                                 (RETURN (LIST (QUOTE EXPT)
                                               X
                                               (QUOTE (QUOTIENT 1 2)) ))))
                          (SETQ Y (ADD1 Y))
                          (GO A) )))))
```

```
DEFINE
(((ASK (LAMBDA (X)
               (AND (NOT (PRIN1 (QUOTE IS)))
                    (NOT (PRIN1 BLANK))
                    (PRINT X)
                    (PRINT (QUOTE POSITIVE))
                    (EQ (RDFLX) (QUOTE YES)) )))))

DEFINE
(((TRIG1 (LAMBDA (A) (MEMBER A (QUOTE (SIN COS)))))
  (SUPERTRIG
   (LAMBDA (EXP)
    (COND
     ((FREE EXP) T)
     ((ATOM EXP) NIL)
     ((MEMBER (CAR EXP) (QUOTE (PLUS TIMES)))
      (AND (SUPERTRIG (CADR EXP))
       (OR (NULL (CDDR EXP)) (SUPERTRIG (CONS (CAR EXP) (CDDR EXP))))))
     ((MEMBER (CAR EXP) (QUOTE (EXPT LOG)))
      (AND (SUPERTRIG (CADR EXP)) (SUPERTRIG (CADDR EXP))) )
     ((MEMBER (CAR EXP) (QUOTE (SIN COS TAN SEC COT CSC)))
      (COND
       ((M2 (CADR EXP) TRIGARG NIL) T)
       ((M2
         (CADR EXP)
         (QUOTE (PLUS (COEFFPT (B FREE) (X VARP)) (COEFFPT (A FREE))))
         NIL )
        (AND (SETQ NOTSAME T) NIL) )
       (T (SUPERTRIG (CADR EXP))) ))
     (T (SUPERTRIG (CADR EXP))) )))
  (SUBST2 (LAMBDA (EXP PAT)
    (COND
     ((NULL EXP) NIL)
     ((M2 EXP PAT NIL) VAR)
     ((ATOM EXP) EXP)
     (T (CONS (SUBST2 (CAR EXP) PAT) (SUBST2 (CDR EXP) PAT))) )))
  (MONSTERTRIG
   (LAMBDA
    (EXP VAR TRIGARG)
    (PROG (NOTSAME W A B Y D)
          (COND
           ((SUPERTRIG EXP) (GO A))
           ((NULL NOTSAME) (RETURN NIL))
           ((NOT (SETQ Y
                   (M2
                    EXP
                    (QUOTE (TIMES
                      (COEFFTT (A FREE))
                      ((B TRIG1) (TIMES (X VARP) (COEFFTT (M FREE))))
                      ((D TRIG1) (TIMES (X VARP) (COEFFTT (N FREE))))))
                    NIL )))
            (GO B) )
           ((NOT (AND
              (MEMBER (SETQ B (CDR (SASSOC (QUOTE B) Y)))
               (QUOTE (SIN COS)) )
              (MEMBER (SETQ D (CDR (SASSOC (QUOTE D) Y)))
               (QUOTE (SIN COS)) )))
            (RETURN NIL) )
           ((AND (EQ B (QUOTE SIN)) (EQ D (QUOTE SIN)))
            (RETURN
             (SIMPTIMES
```

```
          (SUBVAR
           (SUBLIS
            Y
            (QUOTE (A (DIFFERENCE
                (QUOTIENT
                 (SIN (TIMES (DIFFERENCE M N) X))
                 (TIMES 2 (DIFFERENCE M N)) )
                (QUOTIENT (SIN (TIMES (PLUS M N) X))
                 (TIMES 2 (PLUS M N)) )))))))))
       ((AND (EQ B (QUOTE COS)) (EQ D (QUOTE COS)))
        (RETURN
         (SIMPTIMES
          (SUBVAR
           (SUBLIS
            Y
            (QUOTE (A (PLUS
                (QUOTIENT
                 (SIN (TIMES (DIFFERENCE M N) X))
                 (TIMES 2 (DIFFERENCE M N)) )
                (QUOTIENT (SIN (TIMES (PLUS M N) X))
                 (TIMES 2 (PLUS M N)) )))))))))
       ((OR (AND
          (EQ B (QUOTE COS))
          (SETQ W (CDR (SASSOC (QUOTE M) Y)))
          (RPLACD (SASSOC (QUOTE M) Y) (CDR (SASSOC (QUOTE N) Y)))
          (RPLACD (SASSOC (QUOTE N) Y) W) )
         T )
        (RETURN
         (SIMPTIMES
          (SUBVAR
           (SUBLIS
            Y
            (QUOTE (-1 A
              (PLUS
               (QUOTIENT
                (COS (TIMES (DIFFERENCE M N) X))
                (TIMES 2 (DIFFERENCE M N)) )
               (QUOTIENT (COS (TIMES (PLUS M N) X))
                (TIMES 2 (PLUS M N)) )))))))))
B
     (COND
      ((NOT
        (SETQ
         Y
         (PROG2
          (SETQ TRIGARG VAR)
          (M2
           EXP
           (QUOTE (TIMES
             (COEFFTT (A FREE))
             ((B TRIG1) (TIMES (X VARP) (COEFFTT (N INTEGERP))))
             (COEFFTT (C SUPERTRIG)) ))
           NIL ))))
       (RETURN NIL) ))
      (RETURN
       (INTEGRATE
        (EXPAND2
         (LIST
          (QUOTE TIMES)
          (REPLACE Y (QUOTE C))
          (COND
```

```
                ((EQ (SETQ B (REPLACE Y (QUOTE B))) (QUOTE COS))
                 (SUBST
                  VAR
                  (QUOTE X)
                  (SUPERCOSNX (REPLACE Y (QUOTE N))) ))
                (T (SUBST VAR (QUOTE X) (SUPERSINX (REPLACE Y (QUOTE N))))) ))))
            VAR ))
      A
          (SETQ W (SUBST2 EXP TRIGARG))
          (SETQ B
           (CDR (SASSOC (QUOTE B)
             (M2
              TRIGARG
              (QUOTE (PLUS (COEFFPT (B FREE) (X VARP)) (COEFFPT (A FREE))) )
              NIL ))))
          (RETURN (SUBST TRIGARG VAR (TRIGINT (SIMPQUOTIENT (LIST W B)) VAR)) ))))
  (TRIG2 (LAMBDA (A) (MEMBER A (QUOTE (SIN COS TAN COT SEC CSC))))) ))

DEFINE
(((SUPERSINX (LAMBDA (N)
                     ((LAMBDA (I)
                              (EXPAND2 (LIST (QUOTE TIMES)
                                             I
                                             (SINNX (TIMES I N)) )))
                      (COND ((LESSP N 0) -1) (T 1)) )))
  (SUPERCOSNX (LAMBDA (N)
                     ((LAMBDA (I) (EXPAND2 (COSNX (TIMES I N))))
                      (COND ((LESSP N 0) -1) (T 1)) )))
  (SINNX (LAMBDA (N)
                 (COND ((EQUAL N 1) (QUOTE (SIN X)))
                       (T (LIST (QUOTE PLUS)
                                (LIST (QUOTE TIMES)
                                      (QUOTE (SIN X))
                                      (COSNX (SUB1 N)) )
                                (LIST (QUOTE TIMES)
                                      (QUOTE (COS X))
                                      (SINNX (SUB1 N)) ))))))
  (COSNX (LAMBDA (N)
                 (COND ((EQUAL N 1) (QUOTE (COS X)))
                       (T (LIST (QUOTE PLUS)
                                (LIST (QUOTE TIMES)
                                      (QUOTE (COS X))
                                      (COSNX (SUB1 N)) )
                                (LIST (QUOTE TIMES)
                                      -1
                                      (QUOTE (SIN X))
                                      (SINNX (SUB1 N)) )))))))

DEFINE
(((POSEVEN (LAMBDA (A) (AND (EVEN A) (GREATERP A -1))))
  (TRIGFREE (LAMBDA (A)
    (COND
     ((ATOM A) (NOT (MEMBER A (QUOTE (SIN* COS* SEC* TAN*)))))
     (T (AND (TRIGFREE (CAR A)) (TRIGFREE (CDR A)))) )))
  (UNTR (LAMBDA (EXP)
    (COND
     ((ATOM EXP) EXP)
     ((EQ (CAR EXP) (QUOTE LOG))
      (COND ((NULL (CDDR EXP))
             (LIST (CAR EXP) (QUOTE E) (UNTR (CADR EXP))) )
            (T (LIST (CAR EXP) (CADR EXP) (UNTR (CADDR EXP)))) ))
```

```
     ((EQ (CAR EXP) (QUOTE MINUS)) (LIST (QUOTE TIMES) -1 (UNTR (CADR EXP))))
     ((EQ (CAR EXP) (QUOTE SQRT))
      (LIST (QUOTE EXPT) (UNTR (CADR EXP))  0.5E0) )
     ((EQ (CAR EXP) (QUOTE INTEGRAL)) (LIST (CAR EXP) (CADR EXP) VAR))
     ((EQ (CAR EXP) (QUOTE DIFFERENCE))
      (LIST (QUOTE PLUS)
            (UNTR (CADR EXP))
            (LIST (QUOTE TIMES) -1 (UNTR (CADDR EXP))) ))
     ((EQ (CAR EXP) (QUOTE QUOTIENT))
      (LIST (QUOTE TIMES)
            (UNTR (CADR EXP))
            (LIST (QUOTE EXPT) (UNTR (CADDR EXP)) -1) ))
     (T (MAPLIST EXP (FUNCTION (LAMBDA (A) (UNTR (CAR A)))))) )))))

DEFINE
(((TRANSL
   (LAMBDA
    (EXP)
    (COND
     ((NUMBERP EXP)
      (PROG (TEMP)
            (RETURN (COND
              ((FIXP EXP) EXP)
              ((SETQ TEMP (INTEGERP EXP)) TEMP)
              ((SETQ TEMP (DENOMFIND EXP))
               (LIST (QUOTE QUOTIENT) (INTEGERP (TIMES TEMP EXP)) TEMP) )
              (T (ERROR (QUOTE TRANSL))) ))))
     ((ATOM EXP) EXP)
     ((AND (MEMBER (CAR EXP) (QUOTE (PLUS TIMES)))
       (GREATERP (LENGTH (CDR EXP)) 2) )
      (LIST
       (CAR EXP)
       (TRANSL (CADR EXP))
       (TRANSL (CONS (CAR EXP) (CDDR EXP))) ))
     ((AND (EQ (CAR EXP) (QUOTE LOG)) (CDDR EXP))
      (COND ((EQ (CADR EXP) (QUOTE E)) (CONS (CAR EXP) (CDDR EXP)))
       (T (LIST
         (QUOTE QUOTIENT)
         (LIST (QUOTE LOG) (TRANSL (CADDR EXP)))
         (LIST (QUOTE LOG) (CADR EXP)) ))))
     (T (MAPLIST EXP (FUNCTION (LAMBDA (A) (TRANSL (CAR A)))))) )))
  (RAT1 (LAMBDA (EXP)
    (PROG (B1 NOTSAME)
          (COND ((AND (NUMBERP EXP) (ZEROP EXP)) (RETURN NIL)))
          (SETQ B1 (SUBST B (QUOTE B) (QUOTE (EXPT B (N EVEN)))))
          (RETURN (PROG2 (SETQ YY (RAT EXP)) (COND ((NOT NOTSAME) YY) (T NIL))) ))))
  (RAT
   (LAMBDA
    (EXP)
    (PROG (Y)
          (RETURN
           (COND
            ((EQ EXP A) (QUOTE X))
            ((ATOM EXP)
             (COND ((MEMBER EXP (QUOTE (SIN* COS* SEC* TAN*)))
               (SETQ NOTSAME T) )
              (T EXP) ))
            ((SETQ Y (M2 EXP B1 NIL)) (F3 Y))
            (T (CONS (CAR EXP)
              (MAPLIST (CDR EXP) (FUNCTION (LAMBDA (G) (RAT (CAR G))))) ))))))))
  (F3 (LAMBDA (Y)
```

```
     ((EQ (CAR EXP) (QUOTE MINUS)) (LIST (QUOTE TIMES) -1 (UNTR (CADR EXP))))
     ((EQ (CAR EXP) (QUOTE SQRT))
      (LIST (QUOTE EXPT) (UNTR (CADR EXP))  0.5E0) )
     ((EQ (CAR EXP) (QUOTE INTEGRAL)) (LIST (CAR EXP) (CADR EXP) VAR))
     ((EQ (CAR EXP) (QUOTE DIFFERENCE))
      (LIST (QUOTE PLUS)
            (UNTR (CADR EXP))
            (LIST (QUOTE TIMES) -1 (UNTR (CADDR EXP))) ))
     ((EQ (CAR EXP) (QUOTE QUOTIENT))
      (LIST (QUOTE TIMES)
            (UNTR (CADR EXP))
            (LIST (QUOTE EXPT) (UNTR (CADDR EXP)) -1) ))
     (T (MAPLIST EXP (FUNCTION (LAMBDA (A) (UNTR (CAR A)))))) )))))

DEFINE
(((TRANSL
   (LAMBDA
    (EXP)
    (COND
     ((NUMBERP EXP)
      (PROG (TEMP)
            (RETURN (COND
              ((FIXP EXP) EXP)
              ((SETQ TEMP (INTEGERP EXP)) TEMP)
              ((SETQ TEMP (DENOMFIND EXP))
               (LIST (QUOTE QUOTIENT) (INTEGERP (TIMES TEMP EXP)) TEMP) )
             (T (ERROR (QUOTE TRANSL))) ))))
     ((ATOM EXP) EXP)
     ((AND (MEMBER (CAR EXP) (QUOTE (PLUS TIMES)))
       (GREATERP (LENGTH (CDR EXP)) 2) )
      (LIST
       (CAR EXP)
       (TRANSL (CADR EXP))
       (TRANSL (CONS (CAR EXP) (CDDR EXP))) ))
     ((AND (EQ (CAR EXP) (QUOTE LOG)) (CDDR EXP))
      (COND ((EQ (CADR EXP) (QUOTE E)) (CONS (CAR EXP) (CDDR EXP)))
       (T (LIST
          (QUOTE QUOTIENT)
          (LIST (QUOTE LOG) (TRANSL (CADDR EXP)))
          (LIST (QUOTE LOG) (CADR EXP)) ))))
     (T (MAPLIST EXP (FUNCTION (LAMBDA (A) (TRANSL (CAR A)))))) )))
  (RAT1 (LAMBDA (EXP)
    (PROG (B1 NOTSAME)
          (COND ((AND (NUMBERP EXP) (ZEROP EXP)) (RETURN NIL)))
          (SETQ B1 (SUBST B (QUOTE B) (QUOTE (EXPT B (N EVEN)))))
          (RETURN (PROG2 (SETQ YY (RAT EXP)) (COND ((NOT NOTSAME) YY) (T NIL))) ))))
  (RAT
   (LAMBDA
    (EXP)
    (PROG (Y)
          (RETURN
           (COND
            ((EQ EXP A) (QUOTE X))
            ((ATOM EXP)
             (COND ((MEMBER EXP (QUOTE (SIN* COS* SEC* TAN*)))
               (SETQ NOTSAME T) )
              (T EXP) ))
            ((SETQ Y (M2 EXP B1 NIL)) (F3 Y))
            (T (CONS (CAR EXP)
               (MAPLIST (CDR EXP) (FUNCTION (LAMBDA (G) (RAT (CAR G))))) )))))))
  (F3 (LAMBDA (Y)
```

```
   (SUBST
    C
    (QUOTE C)
    (SUBST
     (QUOTIENT (CDR (SASSOC (QUOTE N) Y NIL)) 2)
     (QUOTE N)
     (QUOTE (EXPT (PLUS 1 (TIMES C (EXPT X 2))) N)) ))))
(ODD1
 (LAMBDA
  (N)
  (COND ((NOT (ZEROP (REMAINDER N 2)))
    (SETQ YZ
     (SUBST
      C
      (QUOTE C)
      (LIST
       (QUOTE EXPT)
       (QUOTE (PLUS 1 (TIMES C (EXPT X 2))))
       (QUOTIENT (SUB1 N) 2) ))))
   (T NIL) )))
(EVEN (LAMBDA (A) (AND (NUMBERP A) (INTEGERP (QUOTIENT A 2)))))
(SUBVAR (LAMBDA (B) (SUBST VAR (QUOTE X) B)))
(TRIGINT
 (LAMBDA
  (EXP VAR)
  (PROG (Y REPL Y1 Y2 YY Z M N C YZ A B)
        (SETQ Y2
         (SUBLIS (SUBVAR (QUOTE (((SIN X) . SIN*)
              ((COS X) . COS*)
              ((TAN X) . TAN*)
              ((COT X) EXPT TAN* -1)
              ((SEC X) . SEC*)
              ((CSC X) EXPT SEC* -1) )))
          EXP ))
        (SETQ Y1
         (SETQ Y
          (SIMP (SUBLIS (QUOTE ((TAN* TIMES SIN* (EXPT COS* -1)) (SEC* EXPT COS* -1)))
            Y2 ))))
        (COND ((NULL (SETQ Z
             (M2
              Y
              (QUOTE (TIMES
                (COEFFTT (B TRIGFREE))
                (EXPT SIN* (M POSEVEN))
                (EXPT COS* (N POSEVEN)) ))
              NIL )))
          (GO L1) ))
        (SETQ M (CDR (SASSOC (QUOTE M) Z)))
        (SETQ N (CDR (SASSOC (QUOTE N) Z)))
        (SETQ A
         (INTEGERP (TIMES
            0.5E0
           (COND ((LESSP M N) 1) (T -1))
           (PLUS N (TIMES -1 M)) )))
        (SETQ Z (CONS (CONS (QUOTE A) A) Z))
        (RETURN
         (SIMP
          (LIST
           (QUOTE TIMES)
           (CDR (SASSOC (QUOTE B) Z))
            0.5E0
```

```
        (SUBST
         (LIST (QUOTE TIMES) 2 VAR)
         (QUOTE X)
         (INTEGRATE
          (SIMP
           (COND
            ((LESSP M N)
             (SUBLIS Z
              (QUOTE (TIMES
                (EXPT (TIMES  0.5E0 (SIN X)) M)
                (EXPT (PLUS  0.5E0 (TIMES  0.5E0 (COS X))) A) ))))
            (T (SUBLIS Z
              (QUOTE (TIMES
                (EXPT (TIMES  0.5E0 (SIN X)) N)
                (EXPT (PLUS  0.5E0 (TIMES -0.5E0 (COS X))) A) ))))))
           (QUOTE X) )))))
L1
     (SETQ C -1)
     (SETQ A (QUOTE SIN*))
     (SETQ B (QUOTE COS*))
     (COND ((AND
        (M2 Y (QUOTE (COEFFPT (C RAT1) (EXPT COS* (N ODD1)))) NIL)
        (SETQ REPL (LIST (QUOTE SIN) VAR)) )
       (GO GETOUT) ))
     (SETQ A B)
     (SETQ B (QUOTE SIN*))
     (COND ((AND
        (M2 Y (QUOTE (COEFFPT (C RAT1) (EXPT SIN* (N ODD1)))) NIL)
        (SETQ REPL (LIST (QUOTE COS) VAR)) )
       (GO GET3) ))
     (SETQ Y
      (SIMP (SUBLIS (QUOTE ((SIN* TIMES TAN* (EXPT SEC* -1)) (COS* EXPT SEC* -1)))
        Y2 )))
     (SETQ C 1)
     (SETQ A (QUOTE TAN*))
     (SETQ B (QUOTE SEC*))
     (COND ((AND (RAT1 Y) (SETQ REPL (LIST (QUOTE TAN) VAR))) (GO GET1)) )
     (SETQ A B)
     (SETQ B (QUOTE TAN*))
     (COND ((AND
        (M2 Y (QUOTE (COEFFPT (C RAT1) (EXPT TAN* (N ODD1)))) NIL)
        (SETQ REPL (LIST (QUOTE SEC) VAR)) )
       (GO GETOUT) ))
     (SETQ Y
      (SIMP (SUBLIS (QUOTE ((SIN* TIMES 2 X (EXPT (PLUS 1 (EXPT X 2)) -1))
          (COS*
           TIMES
           (PLUS 1 (TIMES -1 (EXPT X 2)))
           (EXPT (PLUS 1 (EXPT X 2)) -1) )))
        Y1 )))
     (SETQ Y
      (LIST
       (QUOTE TIMES)
       Y
       (QUOTE (TIMES 2 (EXPT (PLUS 1 (EXPT X 2)) -1))) ))
     (SETQ REPL (SUBVAR (QUOTE (QUOTIENT (SIN X) (PLUS 1 (COS X))))))
     (GO GET2)
GET3
     (SETQ Y (LIST (QUOTE TIMES) -1 YY YZ))
     (GO GET2)
GET1
```

```
          (SETQ Y
           (LIST (QUOTE TIMES) (QUOTE (EXPT (PLUS 1 (EXPT X 2)) -1)) YY) )
          (GO GET2)
     GETOUT
          (SETQ Y (LIST (QUOTE TIMES) YY YZ))
     GET2
          (SETQ Y (SIMP Y))
          (RETURN (SIMP (SUBST REPL (QUOTE X) (INTEGRATE Y (QUOTE X))))) )))))

DEFINE
(((ALGORT
   (LAMBDA
    (R1 P1 VAR1)
    (PROG (R OLDS1 OLDREST P VAR PD Q S S1 S2 ANS A1 A2 A3 NUM A M B REST)
          (CSETQ VARLIST (LIST VAR1))
          (NEWVAR R1)
          (NEWVAR P1)
          (SETQ R (REP R1))
          (SETQ P (REP P1))
          (SETQ VAR (REP VAR1))
          (SETQ PD (PFDERIVATIVE P))
          (SETQ Q (DENOMINATORF R))
          (SETQ S1 (NUMERATORF R))
     LOOP
          (COND ((NOT (POLP S1)) (GO A)))
          (SETQ B (LIST (CAR S1)))
          (SETQ S (SIMPOL (CDR S1)))
          (SETQ M (SUB1 (LENGTH S1)))
     B
          (SETQ ANS (PLUSF A ANS))
          (SETQ OLDS1 S1)
          (SETQ OLDREST REST)
          (SETQ A (QUOTIENTF (TIMESF B (POLEXPT VAR M)) (TIMESF PD Q)))
          (SETQ A3 (TIMESF A (PFDERIVATIVE Q)))
          (SETQ A2
           (QUOTIENTF (MINUSF (TIMESF B (POLDERIVATIVE (POLEXPT VAR M))))
            PD ))
          (SETQ A1
           (QUOTIENTF (TIMESF (TIMESF B (POLEXPT VAR M)) (PFDERIVATIVE PD))
            (POLEXPT PD 2) ))
          (SETQ S2 (SEP (PLUSF (PLUSF S REST) (PLUSF A1 (PLUSF A2 A3)))))
          (SETQ S1 (CAR S2))
          (SETQ REST (CDR S2))
          (COND (S1 (GO LOOP)))
          (SETQ REST (SIMPSIMP (TRANS REST)))
          (COND ((AND (NUMBERP REST) (ZEROP REST))
            (RETURN (SIMPSIMP (LIST
                (QUOTE TIMES)
                (TRANS (PLUSF A ANS))
                (LIST (QUOTE EXPT) (QUOTE E) P1) )))))
          (RETURN
           (PLUSSIMP
            (LIST
             (QUOTE PLUS)
             (SIMPSIMP (LIST
               (QUOTE TIMES)
               (TRANS ANS)
               (LIST (QUOTE EXPT) (QUOTE E) P1) ))
             (LIST
              (QUOTE INTEGRAL)
              (LIST
```

```
                (QUOTE TIMES)
                (COND ((NOT OLDREST) (TRANS (QUOTIENTF OLDS1 Q)))
                      (T (TRANS (QUOTIENTF (CDR S2) Q))) )
                (LIST (QUOTE EXPT) (QUOTE E) P1) )))))
     A
          (SETQ B (CONS (LIST (CAAR S1)) (CDR S1)))
          (SETQ S
           (COND ((SETQ S (SIMPOL (CDAR S1))) (CONS S (CDR S1))) (T S)) )
          (SETQ M (SUB1 (LENGTH (CAR S1))))
          (GO B) )))))

DEFINE
(((SEP (LAMBDA (R)
               (PROG (S D N)
                     (COND ((POLP R) (RETURN (CONS R NIL))))
                     (SETQ N (NUMERATORF R))
                     (SETQ D (DENOMINATORF R))
                     (COND ((AND (ONEP (LENGTH N)) (ONEP (LENGTH D)))
                            (RETURN (CONS R NIL)) ))
                     (SETQ S (POLDIVIDE N D))
                     (RETURN (CONS (CAR S) (QUOTIENTF (CDR S) D))) )))))

DEFINE
(((SUPERALGORT (LAMBDA (R P VAR)
                       (PROG NIL
                             (FILEWRITE (QUOTE MANOVE)
                                        (QUOTE ANS)
                                        (ALGORT R P VAR) )
                             (CHAIN (QUOTE ((R MOSES)))) )))))

DEFINE
(((MASTER (LAMBDA (Y)
                  (PROG (FLIST)
                        (CSETQ VARLIST (LIST (CAR Y)))
                        (NEWVAR (CDR Y))
                        (CSETQ REPSWITCH NIL)
                        (FILEWRITE (QUOTE MANOVE)
                                   (QUOTE ANS)
                                   (SIMPSIMP (FPROG (REP (CDR Y)))) )
                        (CHAIN (QUOTE ((R MOSES)))) )))))

DEFINE
(((RAT8PRIME (LAMBDA (C) (AND (RAT8 C) (OR (NOT (NUMBERP C)) (NOT (ZEROP C))))))
  (FIND (LAMBDA (EXP)
    (COND ((ATOM EXP) (MEMBER EXP (QUOTE (LOG INTEGRAL ARCTAN))))
     (T (OR (FIND (CAR EXP)) (FIND (CDR EXP)))) )))
  (RATLOG
   (LAMBDA
    (EXP VAR FORM)
    (PROG (A B C CC D Y Z W)
          (SETQ Y FORM)
          (SETQ B (CDR (SASSOC (QUOTE B) Y)))
          (SETQ C (CDR (SASSOC (QUOTE C) Y)))
          (SETQ Y (INTEGRATE C VAR))
          (COND ((FIND Y) (RETURN NIL)))
          (SETQ D (DIFF1 (CDR (SASSOC (QUOTE A) FORM)) VAR))
          (COND ((EQ (CADAR FORM) (QUOTE ARCSIN)) (GO B)))
     C
          (SETQ Z (INTEGRATE (SIMPTIMES (LIST Y D)) VAR))
     A
          (SETQ D (CDR (SASSOC (QUOTE A) FORM)))
```

```
        (RETURN (SIMP (LIST (QUOTE DIFFERENCE) (LIST (QUOTE TIMES) Y D) Z)) )
    8
        (COND
         ((NOT
           (SETQ
            W
            (M2
             D
             (QUOTE
              (PLUS
               (COEFFPT
                (C TRUE1)
                (EXPT
                 (CC (LAMBDA (CC)
                   (M1 CC
                    (QUOTE (PLUS (COEFFPT (B FREE) (EXPT (X VARP) 2))
                      (COEFFP (A FREE)) )))))
                 (N INTEGERP1) )))))))
          (GO C) ))
        (SETQ CC (CDR (SASSOC (QUOTE CC) W)))
        (SETQ Z (TRIGSQRT (LIST (QUOTE TIMES) Y D) VAR CC W))
        (COND ((NULL Z) (RETURN NIL)))
        (GO A) )))))

DEFINE
(((FIND1 (LAMBDA (Y A)
    (COND
     ((EQ Y A) T)
     ((ATOM Y) NIL)
     (T (OR (FIND1 (CAR Y) A) (FIND1 (CDR Y) A))) )))
  (MAXPARTS
   (LAMBDA
    (A)
    (PROG (Y)
          (SETQ Y 1)
     LOOP
          (SETQ Y
           (MAX Y
            (COND ((EQ (CAR Y) (QUOTE EXPT))
              (COND ((NUMBERP (CADDAR Y))
                (COND ((LESSP (CADDAR Y) 0) (MINUS (CADDAR Y)))
                 (T (CADDAR Y)) ))
               (T 1) ))
             (T 1) )))
          (SETQ A (CDR A))
          (COND ((NULL A) (RETURN Y)))
          (GO LOOP) )))


            INTEGRATION-BY-PARTS


  (PARTS
   (LAMBDA
    (EXP VAR)
    (PROG (A B Y Z W G TOPPART)
          (COND (NOPARTS (RETURN NIL)))
          (COND ((NOT (GET (QUOTE TOP) (QUOTE APVAL)))
            (CSETQ TOP (SETQ TOPPART (GENSYM))) ))
          (SETQ Y
           (M2
```

```
            EXP
            (QUOTE (TIMES (COEFFTT (A FREE)) (COEFFTT (B TRUE))))
            NIL ))
          (SETQ A (CDR (SASSOC (QUOTE A) Y)))
          (SETQ B (CDR (SASSOC (QUOTE B) Y)))
          (COND ((NOT (EQ (CAR B) (QUOTE TIMES))) (RETURN NIL)))
          (COND
           ((NOT (GET (QUOTE MAXPARTS) (QUOTE APVAL)))
            (AND (CSETQ MAXPARTS (TIMES 2 (MAXPARTS B)))
             (CSETQ NUMPARTS 1) ))
           ((AND (CSETQ NUMPARTS (ADD1 NUMPARTS))
             (GREATERP NUMPARTS MAXPARTS) )
            (RETURN NIL) ))
    B
          (SETQ Y (CDR B))
    LOOP
          (CSETQ NOPARTS T)
          (SETQ Z (INTEGRATE (CAR Y) VAR))
          (CSETQ NOPARTS NIL)
          (COND ((FIND1 Z (QUOTE INTEGRAL)) (GO A)))
          (SETQ G (CHOICE (CAR Y) B))
          (SETQ W (INTEGRATE (SIMPTIMES (LIST (DIFF1 G VAR) Z)) VAR))
          (COND ((FIND1 W (QUOTE INTEGRAL)) (GO A)))
          (SETQ
           Y
           (SIMPTIMES (LIST A (SIMPDIFFERENCE (LIST (SIMPTIMES (LIST G Z)) W))) ))
          (RETURN (COND ((EQ TOPPART TOP)
             (PROG23
              (REMPROP (QUOTE TOP) (QUOTE APVAL))
              Y
              (REMPROP (QUOTE MAXPARTS) (QUOTE APVAL)) ))
            (T Y) ))
    A
          (SETQ Y (CDR Y))
          (COND ((NULL Y) (RETURN NIL)))
          (COND ((NOT (EQ TOP TOPPART)) (GO LOOP)))
          (CSETQ MAXPARTS (TIMES 2 (MAXPARTS B)))
          (CSETQ NUMPARTS 1)
          (GO LOOP) )))))

CSET
(NUMPARTS 1)

CSET
(NOPARTS NIL)
```

SOLDIER

```
DEFINE
(((SOL
   (LAMBDA
    (EXP INDVAR DEPVAR)
    (SUBST
     INDVAR
     (QUOTE X)
     (SUBST
      DEPVAR
      (QUOTE Y)
```

```
    (SOLDIER
     (SUBST
      (QUOTE X)
      INDVAR
      (SUBST
       (QUOTE Y)
       DEPVAR
       (SUBST
        (QUOTE DX)
        (INTERN (MKNAM (O.. .CLEARBUFF) (PACK (QUOTE D)) (PACK INDVAR))) )
        (SUBST
         (QUOTE DY)
         (INTERN (MKNAM (OR (CLEARBUFF) (PACK (QUOTE D)) (PACK DEPVAR))) )
         (SUBST
          (QUOTE YPR)
          (INTERN (MKNAM (OR
             (CLEARBUFF)
             (PACK DEPVAR)
             (PACK (QUOTE P))
             (PACK (QUOTE R)) )))
          EXP ))))))))))
(SOLCON
 (LAMBDA
  (EXP INDVAR DEPVAR X Y)
  ((LAMBDA (Z)
    ((LAMBDA (W)
      (COND ((NULL W) NIL)
       (T (LIST
         (QUOTE EQUAL)
         (SIMP (SUBST Y DEPVAR (SUBST X INDVAR W)))
         W ))))
     (COND
      ((NULL Z) NIL)
      ((EQ (CADR Z) (QUOTE CO)) (CADDR Z))
      (T (CADR Z)) )))
   (SOL EXP INDVAR DEPVAR) )))
(SOLDIER
 (LAMBDA
  (EXP)
  (PROG (W EXP1 EXP2)
        (COND
        ((SETQ W
          (M2
           EXP
           (QUOTE (PLUS (COEFFPT (A TRUE) DY) (COEFFPT (B TRUE) DX)))
           NIL ))
         (GO A) )
        ((SETQ W
          (M2
           EXP
           (QUOTE (PLUS (COEFFPT (A TRUE) YPR) (COEFFPT (B TRUE))))
           NIL ))
         NIL )
        (T (RETURN NIL)) )
       (SETQ EXP1 (REPLACE W (QUOTE (PLUS (TIMES A DY) (TIMES B DX)))))
       (SETQ EXP2 EXP)
       (GO B)
   A
       (SETQ EXP2 (REPLACE W (QUOTE (PLUS (TIMES A YPR) B))))
       (SETQ EXP1 EXP)
   B
```

```
          (COND ((TRYSOLD (QUOTE (LINEAR
                SEP
                EXACT
                HOMOGTYPE
                BERNOULLI
                LINEARCOEFF
                ALMOSTLINEAR
                REVERSEVAR
                XNY1 ))
               (QUOTE (EXP1 EXP1 EXP1 EXP1 EXP2 EXP2 EXP1 EXP1 EXP2)) )
              (RETURN W) )
             (T (RETURN NIL)) ))))
  (TRYSOLD (LAMBDA (A B)
    (COND
     ((NULL A) NIL)
     ((SETQ W ((CAR A) (COND ((EQ (CAR B) (QUOTE EXP1)) EXP1) (T EXP2)))) W)
     (T (TRYSOLD (CDR A) (CDR B))) )))))

DEFINE
(((FACTORXY
   (LAMBDA
    (EXP)
    (COND
     ((NOT (EQ (CAR EXP) (QUOTE TIMES))) EXP)
     (T
      (SIMPTIMES
       (MAPLIST
        (CDR EXP)
        (FUNCTION (LAMBDA (EXP)
          (COND
           ((EQ (CAAR EXP) (QUOTE PLUS)) (FACTORXY2 (CAR EXP)))
           ((AND (EQ (CAAR EXP) (QUOTE EXPT))
             (EQ (CAADAR EXP) (QUOTE PLUS)) )
            (SIMPEXPT (LIST (FACTORXY2 (CADAR EXP)) (CADDAR EXP))) )
           (T (CAR EXP)) ))))))))))
  (FACTORXY2
   (LAMBDA
    (EXP)
    (PROG (Z IND RES W)
          (SETQ Z (CDR EXP))
          (SETQ IND (QUOTE X))
     LOOP
          (COND
           ((NOT
             (SETQ
              W
              (M2
               (CAR Z)
               (QUOTE
                (COEFFT
                 (B TRUE)
                 (EXPT
                  (A M1 IND)
                  (N (FUNCTION (LAMBDA (N)
                     (AND (NUMBERP N) (GREATERP N  0.98999999E0)) ))))))
               NIL )))
            (GO NO) ))
          (SETQ RES
           (CONS (REPLACE W (QUOTE (TIMES B (EXPT A (PLUS N -1))))) RES) )
          (COND ((NOT (SETQ Z (CDR Z)))
            (RETURN (SIMPTIMES (LIST IND (SIMPPLUS RES)))) ))
```

```
          (GO LOOP)
     NO
          (COND ((EQ IND (QUOTE Y)) (RETURN EXP)))
          (SETQ IND (QUOTE Y))
          (SETQ Z (CDR EXP))
          (SETQ RES NIL)
          (GO LOOP) )))))

DEFINE
(((SIMPEXPT
   (LAMBDA
    (EXP)
    (PROG (A B)
          (SETQ B (SIMP (CADR EXP)))
          (SETQ A (SIMP (CAR EXP)))
          (COND
           ((EQP A 0) (RETURN 0))
           ((AND
             (EQ (CAR A) (QUOTE EXPT))
             (SETQ B (SIMPTIMES (LIST B (CADDR A))))
             (SETQ A (CADR A))
             NIL )
            NIL )
           ((EQP B 0) (RETURN 1))
           ((EQP B 1) (RETURN A))
           ((EQP A 1) (RETURN 1))
           ((AND (NUMBERP A) (NUMBERP B))
            (RETURN (COND
              ((NOT EXPTIND) (EXPT A B))
              ((AND (FIXP B) (GREATERP B -1)) (EXPT A B))
              (T (LIST (QUOTE EXPT) A B)) )))
           ((EQ (CAR A) (QUOTE TIMES))
            (RETURN (CONS (QUOTE TIMES) (EXPTLOOP (CDR A)))) )
           ((AND EXPTSUM (EQ (CAR B) (QUOTE PLUS)))
            (RETURN
             (CONS
              (QUOTE TIMES)
              (MAPLIST (CDR B)
               (FUNCTION (LAMBDA (C) (SIMPEXPT (LIST A (CAR C))))) ))))
           ((NOT (ATOM B))
            (RETURN
             (PROG (W)
                   (RETURN
                    (COND
                     ((NOT (SETQ W
                                 (M2
                                  B
                                  (QUOTE (PLUS (CCEFFT (C TRUE1)
                                                       (LOG (B1 TRUE) (A TRUE)) )
                                               (COEFFP (E TRUE)) ))
                                  NIL )))
                      (LIST (QUOTE EXPT) A B) )
                     ((NOT (EQUAL A (SUBLIS W (QUOTE B1))))
                      (LIST (QUOTE EXPT) A B) )
                     (T
                      (SIMPTIMES (LIST
                        (SIMPEXPT (LIST (SUBLIS W (QUOTE A))
                                        (SUBLIS W (QUOTE C)) ))
                        (SIMPEXPT (LIST A (SUBLIS W (QUOTE E)))) ))))))))))
          (RETURN (LIST (QUOTE EXPT) A B)) )))
  (EXPTLOOP
```

```
    (LAMBDA
     (A)
     (PROG23
      (CSETQ SIMPIND T)
      (MAPLIST A (FUNCTION (LAMBDA (C) (SIMPEXPT (LIST (CAR C) B)))))
      (CSETQ SIMPIND NIL) )))))

DEFINE
(((LINEAR
   (LAMBDA
    (EXP)
    (PROG (Y Z W)
          (RETURN
           (COND
            ((NOT
              (SETQ
               W
               (M2
                EXP
                (QUOTE
                 (PLUS
                  (COEFFPT (F FREEX (QUOTE Y)) DY)
                  (COEFFPT (A M1
                    (QUOTE (PLUS (COEFFPT (G FREEX (QUOTE Y)) Y)
                      (COEFFPT (H FREEX (QUOTE Y))) )))
                   DX )))
                NIL )))
             (COND ((AND (THEREXNY EXP 1)
                (NOT (M2 EXP (SETQ W (EXPAND2 EXP)) NIL)) )
               (LINEAR W) )
              (T NIL) ))
             (T
              (LIST
               (QUOTE EQUAL)
               (QUOTE CO)
               (SIMPPLUS
                (LIST
                 (LIST
                  (QUOTE TIMES)
                  (QUOTE Y)
                  (SETQ
                   Z
                   (SIMPEXPT
                    (LIST
                     (QUOTE E)
                     (SIN (SIMPQUOTIENT (LIST (REPLACE W (QUOTE G)
                        (REPLACE W (QUOTE F)) ))
                      (QUOTE X) )))))
                 (SIN
                  (SIMPTIMES (LIST Z
                    (SIMPQUOTIENT (LIST (REPLACE W (QUOTE H))
                      (REPLACE W (QUOTE F)) ))))
                  (QUOTE X) ))))))))))
  (THEREXNY (LAMBDA (EXP N) (EQUAL N (COUNTY EXP))))
  (COUNTY (LAMBDA (EXP)
    (COND ((ATOM EXP) (COND ((EQ EXP (QUOTE Y)) 1) (T 0)))
     (T (PLUS (COUNTY (CAR EXP)) (COUNTY (CDR EXP)))) ))))

DEFINE
(((SEP
   (LAMBDA
```

```
    (EXP)
    (PROG (W)
          (RETURN
           (COND
            ((SETQ W
              (M2
               (PROG23 (CSETQ EXPTSUM T) (SIMP EXP) (CSETQ EXPTSUM NIL))
               (QUOTE (PLUS
                 (TIMES
                  DX
                  (COEFFTT (M FREEX (QUOTE X)))
                  (COEFFTT (R FREEX (QUOTE Y))) )
                 (TIMES
                  DY
                  (COEFFTT (N FREEX (QUOTE X)))
                  (COEFFTT (S FREEX (QUOTE Y))) )))
               NIL ))
             (LIST
              (QUOTE EQUAL)
              (SIMPPLUS (LIST
                (SIN (SUBLIS W (QUOTE (QUOTIENT R S))) (QUOTE X))
                (SIN (SUBLIS W (QUOTE (QUOTIENT N M))) (QUOTE Y)) ))
              (QUOTE CO) ))
             (T NIL) )))))
  (FREEX (LAMBDA (A VAR)
    (COND ((ATOM A) (NOT (EQ A VAR)))
     (T (AND (FREEX (CAR A) VAR) (FREEX (CDR A) VAR))) )))))

DEFINE
(((EXACT
   (LAMBDA
    (EXP)
    (PROG (W P Q DPDY DQDX Y F1)
          (COND ((NOT (SETQ W
               (M2
                EXP
                (QUOTE (PLUS (COEFFPT (P TRUE) DX) (COEFFPT (Q TRUE) DY)) )
                NIL )))
            (RETURN NIL) ))
          (SETQ P (SUBLIS W (QUOTE P)))
          (SETQ Q (SUBLIS W (QUOTE Q)))
          (SETQ DPDY (DIFF1 P (QUOTE Y)))
          (SETQ DQDX (DIFF1 Q (QUOTE X)))
          (COND ((NOT (M2 DPDY DQDX NIL)) (GO A)))
     OUT
          (SETQ Y (SIN P (QUOTE X)))
          (RETURN
           (LIST
            (QUOTE EQUAL)
            (QUOTE CO)
            (SIMPPLUS
             (LIST
              Y
              (SIN
               (EXPAND2 (SIMPDIFFERENCE (LIST Q (DIFF1 Y (QUOTE Y)))))
               (QUOTE Y) )))))
     A
          (COND
           ((NOT
             (FREEX
              (SETQ F1
```

```
                (SIMPQUOTIENT (LIST (SIMPDIFFERENCE (LIST DPDY DQDX)) Q)) )
               (QUOTE Y) ))
             (GO B) ))
           (SETQ Y (SIMPEXPT (LIST (QUOTE E) (SIN F1 (QUOTE X)))))
           (SETQ P (SIMPTIMES (LIST Y P)))
           (SETQ Q (SIMPTIMES (LIST Y Q)))
           (GO OUT)
      B
           (COND
            ((NOT
              (FREEX
               (SETQ F1
                (SIMPQUOTIENT (LIST (SIMPDIFFERENCE (LIST DQDX DPDY)) P)) )
               (QUOTE X) ))
             (GO C) ))
           (SETQ Y (SIMPEXPT (LIST (QUOTE E) (SIN F1 (QUOTE Y)))))
           (SETQ P (SIMPTIMES (LIST Y P)))
           (SETQ Q (SIMPTIMES (LIST Y Q)))
           (GO OUT)
      C
           (COND ((NOT (AND (M2 DPDY (SIMPMINUS (LIST DQDX)) NIL)
               (M2 (DIFF1 P (QUOTE X)) (DIFF1 Q (QUOTE Y)) NIL) ))
             (RETURN NIL) ))
           (SETQ Y
            (SIMPPLUS (LIST (SIMPTIMES (LIST P P)) (SIMPTIMES (LIST Q Q)))) )
           (SETQ P (SIMPQUOTIENT (LIST P Y)))
           (SETQ Q (SIMPQUOTIENT (LIST Q Y)))
           (GO OUT) )))))

DEFINE
(((BERNOULLI
   (LAMBDA
    (EXP)
    (PROG (W)
          (RETURN
           (COND
            ((NOT
              (SETQ
               W
               (M2
                EXP
                (QUOTE
                 (PLUS
                  (COEFFPT (B TRUE) YPR)
                  (COEFFPT (P FREEX (QUOTE Y)) Y)
                  (COEFFPT
                   (Q FREEX (QUOTE Y))
                   (EXPT Y
                    (N (LAMBDA (A)
                      (AND (NUMBERP A) (NOT (ZEROP A))) ))))))
                NIL )))
             (COND ((AND (THEREXNY EXP 2)
                 (NOT (M2 EXP (SETQ W (EXPAND2 EXP)) NIL)) )
                (BERNOULLI W) )
               (T NIL) ))
             ((FREEX (REPLACE W (QUOTE B)) (QUOTE Y))
              ((LAMBDA
                (P Q N1)
                (SUBST
                 (SIMPEXPT (LIST (QUOTE Y) N1))
                 (QUOTE Y)
```

```
                (LINEAR (SIMPPLUS (LIST (QUOTE DY)
                   (LIST
                    (QUOTE TIMES)
                    (QUOTE DX)
                    (LIST
                     (QUOTE PLUS)
                     (LIST (QUOTE TIMES) N1 P (QUOTE Y))
                     (LIST (QUOTE TIMES) N1 Q) ))))))
              (REPLACE W (QUOTE (QUOTIENT P B)))
              (REPLACE W (QUOTE (QUOTIENT Q B)))
              (SIMPDIFFERENCE (LIST 1 (SUBLIS W (QUOTE N)))) )))))))))

DEFINE
(((HOMOGTYPE
   (LAMBDA
    (EXP)
    (PROG (Y Z W)
          (COND
           ((NOT (SETQ W
                  (M2
                   EXP
                   (QUOTE (PLUS (COEFFPT (P TRUE) DX) (COEFFPT (Q TRUE) DY)) )
                   NIL )))
            (RETURN NIL) )
           ((NOT (AND
                  (SETQ Z (HOMOG (SUBLIS W (QUOTE P))))
                  (SETQ Y (HOMOG (SUBLIS W (QUOTE Q))))
                  (EQP Y Z) ))
            (RETURN NIL) )
           (T
            (RETURN
             (LIST
              (QUOTE EQUAL)
              (LIST
               (QUOTE PLUS)
               (QUOTE (LOG E X))
               (SIMP
                (SUBST
                 (QUOTE (QUOTIENT Y X))
                 (QUOTE Y)
                 (SIN
                  (SIMPQUOTIENT
                   (LIST
                    (SETQ Y
                     (SIMP (SUBST 1 (QUOTE X) (SUBLIS W (QUOTE Q))) ))
                    (SIMPPLUS (LIST
                      (SIMP (SUBST 1 (QUOTE X) (SUBLIS W (QUOTE P))) )
                      (SIMPTIMES (LIST (QUOTE Y) Y)) ))))
                   (QUOTE Y) ))))
              (QUOTE CO) )))))))
  (HOMOG (LAMBDA (EXP)
    (PROG (NOTHOM Y)
          (SETQ Y (HOMOGEN EXP))
          (COND (NOTHOM (RETURN NIL)) (T (RETURN Y))) )))
  (HOMOGEN
   (LAMBDA
    (EXP)
    (COND
     ((ATOM EXP) (COND ((EQ EXP (QUOTE Y)) 1) ((EQ EXP (QUOTE X)) 1) (T 0)))
     ((EQ (CAR EXP) (QUOTE TIMES))
      (EVAL (CONS (QUOTE PLUS)
```

```
        (MAPLIST (CDR EXP) (FUNCTION (LAMBDA (C) (HOMOGEN (CAR C))))) )
       (ALIST) ))
     ((EQ (CAR EXP) (QUOTE PLUS))
      ((LAMBDA (Y)
        (PROG (Z)
              (SETQ Z (HOMOGEN (CAR Y)))
         LOOP
              (SETQ Y (CDR Y))
              (COND
               ((NULL Y) (RETURN Z))
               ((NOT (EQUAL Z (HOMOGEN (CAR Y))))
                (RETURN (PROG2 (SETQ NOTHOM T) -1000)) )
               (T (GO LOOP)) )))
        (CDR EXP) ))
     ((EQ (CAR EXP) (QUOTE EXPT))
      (COND
       ((NUMBERP (CADDR EXP)) (TIMES (HOMOGEN (CADR EXP)) (CADDR EXP)))
       ((AND (ZEROP (HOMOGEN (CADR EXP))) (ZEROP (HOMOGEN (CADDR EXP)))) 0)
       (T (PROG2 (SETQ NOTHOM T) -1000)) ))
     ((EQ (CAR EXP) (QUOTE LOG))
      (COND ((ZEROP (HOMOGEN (CADDR EXP))) 0)
       (T (PROG2 (SETQ NOTHOM T) -1000)) ))
     ((ZEROP (HOMOGEN (CADR EXP))) 0)
     (T (PROG2 (SETQ NOTHOM T) -1000)) )))))

DEFINE
(((ALMOSTLINEAR
   (LAMBDA
    (EXP)
    (PROG (W D DDDY)
          (RETURN
           (COND
            ((NULL
              (SETQ
               W
               (M2
                EXP
                (QUOTE
                 (PLUS
                  (TIMES DY (COEFFTT (A TRUE)))
                  (TIMES
                   DX
                   (PLUS
                    (TIMES
                     (COEFFTT (C FREEX (QUOTE Y)))
                     (COEFFTT
                      (D (FUNCTION (LAMBDA (A) (NOT (FREEX A (QUOTE Y))))) ))))
                    (COEFFPP (E FREEX (QUOTE Y))) ))))
                NIL )))
             NIL )
            ((EQUAL 0
              (SETQ DDDY
               (DIFF1 (SETQ D (REPLACE W (QUOTE D))) (QUOTE Y)) ))
             NIL )
            ((NOT (EQUAL 0 (DIFF1 DDDY (QUOTE X)))) NIL)
            (T
             (SUBST
              D
              (QUOTE Y)
              (LINEAR
               (REPLACE
```

```
                (CONS (CONS (QUOTE B)
                  (SIMPQUOTIENT (LIST (REPLACE W (QUOTE A)) DDDY)) )
                 W )
                (QUOTE
                 (PLUS
                  (TIMES B DY)
                  (TIMES DX
                   (PLUS
                    E
                    (TIMES C Y)
                    (TIMES -1 B (EVAL (DIFF1 (QUOTE D) (QUOTE X)))) ))))))))))))))))

DEFINE
(((ZEROP1 (LAMBDA (A) (AND (NUMBERP A) (ZEROP A))))
  (FREEXY (LAMBDA (A) (AND (FREEX A (QUOTE X)) (FREEX A (QUOTE Y)))))
  (LINEARCOEFF
   (LAMBDA
    (EXP)
    (PROG (IND W A B APR BPR)
          (RETURN
           (COND
            ((NOT (ELEMLIN EXP)) NIL)
            ((OR
              (AND
               (ZEROP1 (SETQ A (SUBLIS W (QUOTE A))))
               (ZEROP1 (SETQ B (SUBLIS W (QUOTE B)))) )
              (AND
               (ZEROP1 (SETQ APR (SUBLIS W (QUOTE APR))))
               (ZEROP1 (SETQ BPR (SUBLIS W (QUOTE BPR)))) )
              (ZEROP1 (SIMPDIFFERENCE (LIST (SIMPTIMES (LIST A BPR))
                 (SIMPTIMES (LIST APR B)) ))))
             (RETURN NIL) )
            (T
             (REPLACE
              (REPLACE
               NIL
               (QUOTE
                ((X
                  EVAL
                  (QUOTE*
                   (REPLACE
                    W
                    (QUOTE (PLUS X
                      (MINUS (QUOTIENT
                        (DIFFERENCE (TIMES BPR C)
                         (TIMES B CPR) )
                        (DIFFERENCE (TIMES APR B)
                         (TIMES A BPR) ))))))))
                 (Y
                  EVAL
                  (QUOTE*
                   (REPLACE
                    W
                    (QUOTE (PLUS Y
                      (MINUS (QUOTIENT
                        (DIFFERENCE (TIMES A CPR)
                         (TIMES APR C) )
                        (DIFFERENCE (TIMES APR B)
                         (TIMES A BPR) ))))))))))
              (HOMOGTYPE (SUBSTLIN EXP)) ))))))
  (ELEMLIN
```

```
(LAMBDA
 (EXP)
 ((LAMBDA (Y)
    (COND ((NULL Y) NIL)
     (T (ELEMLIN1 (REPLACE Y (QUOTE (QUOTIENT A B))))) ))
  (M2 EXP (QUOTE (PLUS (COEFFPT (B TRUE) YPR) (COEFFPT (A TRUE))))) NIL) )))
(SUBSTLIN
 (LAMBDA
  (EXP)
  (LIST
   (QUOTE PLUS)
   (QUOTE DY)
   (SIMPTIMES
    (LIST
     (QUOTE DX)
     (SUBSTLIN1 (REPLACE (M2
        EXP
        (QUOTE (PLUS (COEFFPT (B TRUE) YPR) (COEFFPT (A TRUE))))
        NIL )
       (QUOTE (QUOTIENT A B)) )))))))
(ELEMLIN1
 (LAMBDA
  (EXP)
  (COND
   ((FREEXY EXP) T)
   ((SETQ
     W
     (M2
      EXP
      (COND
       (IND IND)
       (T
        (QUOTE (TIMES
          (COEFFTT (AA FREEXY))
          (EXPT (PLUS
            (COEFFPT (A FREEXY) X)
            (COEFFPT (B FREEXY) Y)
            (C FREEXY) )
           (N NUMBERP) )
          (EXPT
           (PLUS
            (COEFFPT (APR FREEXY) X)
            (COEFFPT (BPR FREEXY) Y)
            (CPR FREEXY) )
           (M (FUNCTION (LAMBDA (M N) (EQUAL M (MINUS N)))) N) )))))
     NIL ))
    (COND (IND IND) (T (SETQ IND EXP))) )
   ((ATOM EXP) NIL)
   (T (AND (ELEMLIN1 (CAR EXP)) (ELEMLIN1 (CDR EXP)))) )))
(SUBSTLIN1
 (LAMBDA
  (EXP)
  (COND
   ((FREEXY EXP) T)
   ((M2 EXP IND)
    (SIMP (SUBLIS W
      (QUOTE (TIMES
        AA
        (EXPT (PLUS (TIMES A X) (TIMES B Y)) N)
        (EXPT (PLUS (TIMES APR X) (TIMES BPR Y)) (MINUS N)) )))))
   (T (MAPLIST EXP (FUNCTION (LAMBDA (C) (SUBSTLIN1 (CAR C))))) )))
```

```
DEFINE
(((XNY1
   (LAMBDA
    (EXP)
    (PROG (W C H FX S A B N)
          (COND ((NOT (SETQ W
                      (M2
                       EXP
                       (QUOTE (PLUS (COEFFPT (A TRUE) YPR) (COEFFPT (B TRUE))))
                       NIL )))
            (RETURN NIL) ))
          (SETQ C (REPLACE W (QUOTE (QUOTIENT (MINUS B) A))))
          (SETQ
           H
           (COND
            ((EQ (CAR C) (QUOTE PLUS))
             (SIMPPLUS
              (MAPLIST
               (CDR C)
               (FUNCTION (LAMBDA (G)
                 (SIMPTIMES (LIST (QUOTE X) (QUOTE (EXPT Y -1)) (CAR G)) ))))))
            (T (SIMPTIMES (LIST (QUOTE X) (QUOTE (EXPT Y -1)) C))) ))
          (SETQ FX (QUOTE (TIMES (EXPT X N) Y)))
          (SETQ H (FACTORXY H))
          (SETQ
           S
           (EXPAND2
            (SIMPDIFFERENCE (LIST
              (SIMPTIMES (LIST (DIFF1 H (QUOTE X)) (DIFF1 FX (QUOTE Y))))
              (SIMPTIMES (LIST (DIFF1 H (QUOTE Y)) (DIFF1 FX (QUOTE X))))) ))))
          (COND ((NOT (SETQ W
                      (M2
                       S
                       (QUOTE (PLUS (COEFFPT (A TRUE) N) (COEFFP (B TRUE))))
                       NIL )))
            (RETURN NIL) ))
          (SETQ A (CDR (SASSOC (QUOTE A) W)))
          (SETQ B (CDR (SASSOC (QUOTE B) W)))
          (COND ((OR (ZEROP1 A) (ZEROP1 B)) (RETURN NIL)))
          (SETQ N
           (COND
            ((AND (EQ (CAR A) (QUOTE PLUS)) (EQ (CAR B) (QUOTE PLUS)))
             (MATCHSUM (CDR (SIMPMINUS (LIST B))) (CDR A)) )
            (T (SIMPQUOTIENT (LIST (SIMPMINUS (LIST B)) A))) ))
          (COND ((NOT (NUMBERP N)) (RETURN NIL)))
          (RETURN
           (LIST
            (QUOTE EQUAL)
            (QUOTE C0)
            (SIMPQUOTIENT
             (LIST
              (SIMPEXPT
               (LIST
                (QUOTE E)
                (REPLACE
                 (LIST (CONS
                   (QUOTE U)
                   (SIMPTIMES (LIST (QUOTE Y) (SIMPEXPT (LIST (QUOTE X) N))) )))
                 (SIN
                  (LIST
```

```
                    (QUOTE QUOTIENT)
                    1
                    (LIST
                     (QUOTE TIMES)
                     (QUOTE U)
                     (LIST
                      (QUOTE PLUS)
                      N
                      (REPLACE
                       (LIST (CONS (QUOTE Y)
                          (SIMP (LIST
                             (QUOTE QUOTIENT)
                             (QUOTE U)
                             (LIST (QUOTE EXPT) (QUOTE X) N) ))))
                        H ))))
                   (QUOTE U) ))))
               (QUOTE X) )))))))))


                ADDITIONAL METHODS


DEFINE
(((REVERSEVAR
   (LAMBDA
    (EXP)
    (PROG (Y)
          (RETURN (COND ((SETQ Y
              (LINEAR (SUBLIS (QUOTE ((X . Y) (Y . X) (DX . DY) (DY . DX)))
                EXP )))
             (SUBLIS (QUOTE ((X . Y) (Y . X))) Y) )
            (T NIL) )))))))

DEFINE
(((XAYB
   (LAMBDA
    (EXP)
    (PROG (W
           M
           N
           XYDMDY
           XYDNDX
           XM
           YN
           COEXM
           COEYN
           XAYB
           A
           B
           FORM
           XYDIFF
           A1
           A2
           B1
           B2
           C1
           C2
           DET
           FACT )
          (COND ((NOT (SETQ W
```

```
        (M2
         EXP
         (QUOTE (PLUS (COEFFPT (M TRUE) DX) (COEFFPT (N TRUE) DY)) )
         NIL )))
       (RETURN NIL) ))
     (SETQ M (REPLACE W (QUOTE M)))
     (SETQ N (REPLACE W (QUOTE N)))
     (SETQ XYDMDY
      (EXPAND2 (SIMPTIMES (LIST (QUOTE X) (QUOTE Y) (DIFF1 M (QUOTE Y)))) ))
     (SETQ XYDNDX
      (EXPAND2 (SIMPTIMES (LIST (QUOTE X) (QUOTE Y) (DIFF1 N (QUOTE X)))) ))
     (SETQ XM (EXPAND2 (SIMPTIMES (LIST (QUOTE X) M))))
     (SETQ YN (EXPAND2 (SIMPTIMES (LIST -1 (QUOTE Y) N))))
     (SETQ XYDIFF (SIMPDIFFERENCE (LIST XYDNDX XYDMDY)))
     (SETQ W
      (M2
       (COND ((EQ (CAR YN) (QUOTE PLUS)) (CADR YN)) (T YN))
       (QUOTE (TIMES (COEFFTT (B FREEXY)) (COEFFTT (C TRUE))))
       NIL ))
     (SETQ B1 (REPLACE W (QUOTE B)))
     (SETQ FACT (REPLACE W (QUOTE C)))
     (SETQ YN
      (COND ((EQ (CAR YN) (QUOTE PLUS)) (CONS (QUOTE PLUS) (CDDR YN)))
       (T 0) ))
     (SETQ FORM
      (LIST
       (QUOTE PLUS)
       (CONS (QUOTE COEFFPT)
        (CONS (QUOTE (B FREEXY))
         (COND ((EQ (CAR FACT) (QUOTE TIMES)) (CDR FACT))
          (T (LIST FACT)) )))
       (QUOTE (COEFFPP (D TRUE))) ))
     (SETQ W (M2 XM FORM NIL))
     (SETQ A1 (REPLACE W (QUOTE B)))
     (SETQ XM (REPLACE W (QUOTE D)))
     (SETQ W (M2 XYDIFF FORM NIL))
     (SETQ C1 (REPLACE W (QUOTE B)))
     (SETQ XYDIFF (REPLACE W (QUOTE D)))
     (COND ((M2 YN 0 NIL) (GO B2ZERO)))
     (SETQ W
      (M2
       (COND ((EQ (CAR YN) (QUOTE PLUS)) (CADR YN)) (T YN))
       (QUOTE (TIMES (COEFFTT (B FREEXY)) (COEFFTT (C TRUE))))
       NIL ))
     (SETQ B2 (REPLACE W (QUOTE B)))
     (SETQ FACT (REPLACE W (QUOTE C)))
     (SETQ FORM
      (LIST
       (QUOTE PLUS)
       (CONS (QUOTE COEFFPT)
        (CONS (QUOTE (B FREEXY))
         (COND ((EQ (CAR FACT) (QUOTE TIMES)) (CDR FACT))
          (T (LIST FACT)) )))
       (QUOTE (COEFFPP (D TRUE))) ))
     (SETQ W (M2 XM FORM NIL))
     (SETQ A2 (REPLACE W (QUOTE B)))
B2BACK
     (SETQ W (M2 XYDIFF FORM NIL))
     (SETQ C2 (REPLACE W (QUOTE B)))
     (SETQ DET
      (SIMP (LIST
```

```
            (QUOTE DIFFERENCE)
            (LIST (QUOTE TIMES) B2 A1)
            (LIST (QUOTE TIMES) B1 A2) )))
          (COND ((M2 DET 0 NIL) (RETURN NIL)))
          (SETQ B
           (SIMP (LIST
            (QUOTE QUOTIENT)
            (LIST
             (QUOTE DIFFERENCE)
             (LIST (QUOTE TIMES) B2 C1)
             (LIST (QUOTE TIMES) B1 C2) )
            DET )))
          (SETQ A
           (SIMP (LIST
            (QUOTE QUOTIENT)
            (LIST
             (QUOTE DIFFERENCE)
             (LIST (QUOTE TIMES) A1 C2)
             (LIST (QUOTE TIMES) A2 C1) )
            DET )))
          (SETQ XAYB
           (SIMPTIMES (LIST (LIST (QUOTE EXPT) (QUOTE X) A)
            (LIST (QUOTE EXPT) (QUOTE Y) B) )))
          (RETURN (EXACT (LIST
            (QUOTE PLUS)
            (LIST
             (QUOTE TIMES)
             (QUOTE DX)
             (EXPAND2 (SIMPTIMES (LIST M XAYB))) )
            (LIST
             (QUOTE TIMES)
             (QUOTE DY)
             (EXPAND2 (SIMPTIMES (LIST N XAYB))) ))))
     B2ZERO
          (SETQ B2 0)
          (SETQ W
           (M2
            (COND ((EQ (CAR XM) (QUOTE PLUS)) (CADR XM)) (T XM))
            (QUOTE (TIMES (COEFFTT (B FREEXY)) (COEFFTT (C TRUE))))
            NIL ))
          (SETQ A2 (REPLACE W (QUOTE B)))
          (SETQ FACT (REPLACE W (QUOTE C)))
          (SETQ FORM
           (LIST
            (QUOTE PLUS)
            (CONS (QUOTE COEFFPT)
             (CONS (QUOTE (B FREEXY))
              (COND ((EQ (CAR FACT) (QUOTE TIMES)) (CDR FACT))
               (T (LIST FACT)) )))
            (QUOTE (COEFFPP (D TRUE))) ))
          (GO B2BACK) )))))

DEFINE
(((KAMKE329
   (LAMBDA
    (EXP)
    (PROG (W DET AA BB)
          (COND
           ((NOT
             (SETQ
              W
```

```
            (M2
             (EXPAND2 EXP)
             (QUOTE
              (PLUS
               (COEFFPT (C M1
                 (QUOTE (PLUS (COEFFPT (ALPHA FREEXY) X)
                   (COEFFPT
                    (A FREEXY)
                    (EXPT X (P FREEXY))
                    (EXPT Y (Q FREEXY)) ))))
                YPR )
               (COEFFPT (BETA FREEXY) Y)
               (COEFFPT
                (B FREEXY)
                (EXPT X (R FREEXY))
                (EXPT Y (S FREEXY)) )))
              NIL )))
           (RETURN NIL) )
          ((NOT (AND
             (M2 1 (REPLACE W (QUOTE (DIFFERENCE P R))) NIL)
             (M2 1 (REPLACE W (QUOTE (DIFFERENCE S Q))) NIL) ))
           (RETURN NIL) )
          ((M2
            0
            (SETQ DET
             (REPLACE W
              (QUOTE (DIFFERENCE (TIMES A BETA) (TIMES B ALPHA))) ))
            NIL )
           (RETURN NIL) ))
         (SETQ AA
          (REPLACE W
           (QUOTE (QUOTIENT (DIFFERENCE (TIMES Q BETA) (TIMES R ALPHA))
             (EVAL DET) ))))
         (SETQ BB
          (REPLACE W
           (QUOTE (QUOTIENT (DIFFERENCE (TIMES Q B) (TIMES R A)) (EVAL DET)) )))
         (RETURN
          (REPLACE
           W
           (QUOTE
            (EQUAL CO
             (PLUS
              (QUOTIENT (TIMES (EXPT Y (TIMES A (EVAL AA)))
                (EXPT X (TIMES B (EVAL AA))) )
               (EVAL AA) )
              (QUOTIENT (TIMES (EXPT Y (TIMES ALPHA (EVAL BB)))
                (EXPT X (TIMES BETA (EVAL BB))) )
               (EVAL BB) ))))))))))


                EDGE


DEFINE((
(FREE(LAMBDA(A)(COND((ATOM A)(NOT(EQ A VAR)))
(T(AND(FREE(CAR A))(FREE (CDR A))))))  ))
DEFINE((
(EDGE(LAMBDA(EXP VAR)(PROG
(PROBL ARCLOG POSEXPT OLDPROBL ONEMORE NONRAT NEWB' G W CONST NONCON
 B ANSW L FF AORA' H A
```

```
NINTXP A' B' LDERIV M)
(SETQ B' (TRIGSUBST EXP))
(SETQ NINTXP(M2 B'(QUOTE(TIMES(BB M1(QUOTE(EXPT(A (QUOTE(LAMBDA(X)(NOT
   (FREE X)))))(N
  (QUOTE(LAMBDA(X)(NOT(NUMBERP X)))) ))))
  (COEFFTT(C TRUE))))NIL))
(GO BEG)


LOOP(COND((RAT8 B')(GO FINISHED)))


(COND((EQ(CAR NONCON)(QUOTE TIMES))(GO AA)))
(SETQ FF NONCON)
(GO GUESS)
AA(SETQ LDERIV(CONS(QUOTE PLUS)(MAPLIST (CDR NONCON)
  (FUNCTION(LAMBDA(C)(DIFF1(CAR C)VAR))))))
(SETQ M(CDR NONCON))
(SETQ L(CDR LDERIV))
LOOP2(COND((RAT8(CAR M))(GO SKIP)))
(COND((NOT(M2 (CHOICE (CAR L) LDERIV)
  (LIST(QUOTE PLUS)(LIST(QUOTE TIMES)(CAR M)
  (QUOTE(COEFFTT(A TRUE))))
   (QUOTE(B TRUE))) NIL)) (GO ENDP)))
SKIP(SETQ NONRAT(CAR M))
(SETQ M(CDR M))
(SETQ L(CDR L))
(COND(M(GO LOOP2)))
(SETQ FF NONRAT)
(GO GUESS)
ENDP(SETQ FF(CAR M))


GUESS(SETQ ARCLOG NIL)
(SETQ POSEXPT NIL)
(SETQ G(COND
((EQ(CAR FF)(QUOTE COS))(PROG2(SETQ AORA' T)(LIST(QUOTE SIN)(CADR FF))))
((EQ(CAR FF)(QUOTE SIN))(PROG2(SETQ AORA' T)(LIST(QUOTE COS)(CADR FF))))
((EQ(CAR FF)(QUOTE LOG))(PROG2(SETQ AORA' NIL)FF))
((EQ(CAR FF)(QUOTE ARCSIN))(PROG2(SETQ AORA' NIL)FF))
((EQ(CAR FF)(QUOTE ARCTAN))(PROG2(SETQ AORA' NIL)FF))
((EQ(CAR FF)(QUOTE EXPT))(COND
((FREE(CADR FF))(PROG2(SETQ AORA' T)FF))
((NOT(NUMBERP(CADDR FF)))(PROG23(SETQ AORA' T)(LIST(QUOTE EXPT)(CADR FF)
 (SIMPPLUS(LIST(CADDR FF)1))))
  (SETQ POSEXPT T)))


((GREATERP(CADDR FF)0)(PROG23(SETQ AORA' T)(LIST(QUOTE EXPT)(CADR FF)
  (SIMPPLUS(LIST(CADDR FF)1)))
  (SETQ POSEXPT T)))
((LESSP(CADDR FF)-1)(PROG2(SETQ AORA' T)(LIST(QUOTE EXPT)
  (CADR FF)(SIMPPLUS(LIST(CADDR FF)1)))))
((AND(EQUAL(CADDR FF)-0.5)(SETQ W(M2(CADR FF)
 (QUOTE(PLUS(COEFFP(A FREE0))(COEFFT(C M2(QUOTE(EXPT(D TRUE)(N EVEN)))NIL)
  (B FREE))))NIL)))
 (PROG23(SETQ AORA' T)(REPLACE W
 (QUOTE(ARCSIN(EXPT(QUOTIENT(TIMES(MINUS B)C)A)0.5))))(SETQ ARCLOG T)))
((EQUAL(CADDR FF)-1)(COND((SETQ W(M2(CADR FF)
 (QUOTE(PLUS(COEFFP(A FREE0))(COEFFT(C M2(QUOTE(EXPT(D TRUE)(N EVEN)))NIL)
   (B FREE))))NIL))
```

```
 (PROG23(SETQ AORA' T)(REPLACE W(QUOTE(ARCTAN(EXPT(QUOTIENT(TIMES B C)A)0.5))))
  (SETQ ARCLOG T) ))
  (T(PROG23(SETQ AORA' T)(LIST(QUOTE LOG)(QUOTE E)(CADR FF))(SETQ ARCLOG T)))))
(T(ERROR(QUOTE(NOT YET ACCOUNTED FOR))))))
(T(ERROR(QUOTE(GUESS NOT YET FINISHED))))))


GOGO(COND((NOT AORA')(GO A'SET)))
(SETQ A(SIMPQUOTIENT(LIST NONCON(DIFF1 G VAR))))
(SETQ A(COND((AND ARCLOG(SETQ W(M2 A(QUOTE
  (TIMES(B M2(QUOTE(EXPT(PLUS(COEFFP(B1 FREE0))
   (COEFFT(B2 TRUE)(B3 FREE)))-1))NIL)
  (C M2(QUOTE(PLUS(COEFFP(C1 FREE0))
  (COEFFT(C2 TRUE1)(C3 FREE))))NIL)(COEFFTT(D TRUE))))NIL)))
  (COND((SETQ M(MATCHSUM(CDADR(REPLACE W(QUOTE B)))
   (CDR(REPLACE W(QUOTE C)))))
(SIMPQUOTIENT(LIST(REPLACE W(QUOTE D))M)))
  (T A)))
 (T A)))
(SETQ A'(DIFF1 A VAR))
(SETQ NEWB'(COND((NOT(EQ(CAR A')(QUOTE PLUS)))(SIMPMINUS(LIST
   (SIMPTIMES(LIST G A')))))
 (T(TIMESLOOP(SIMPMINUS(LIST G))(CDR A)))))
(GO LOOP5)
A'SET(SETQ A'(SIMPQUOTIENT(LIST NONCON G)))
(COND((FIND1(SETQ A(INTEGRATE A' VAR))(QUOTE INTEGRAL))(GO KILL)))
(SETQ NEWB'(COND((EQ(CAR A)(QUOTE PLUS))(TIMESLOOP(SIMPMINUS
    (LIST(DIFF1 G VAR)))(CDR A)))
 (T(SIMPTIMES(LIST -1(DIFF1 G VAR)A)))))


LOOP5(SETQ PROBL(CONS(LIST B' CONST NONCON G FF A A' ARCLOG POSEXPT)PROBL))
(COND((AND ARCLOG(NOT(FREE A)))(SETQ ARCLOG 1)))
(COND((AND POSEXPT(NOT(FREE A)))(SETQ POSEXPT 1)))
(PRINT NEWB')
(SETQ B' NEWB')
BEG (SETQ W(M2 B'(QUOTE(TIMES(COEFFTT(A FREE))(COEFFTT(B TRUE))))NIL))
(SETQ CONST(REPLACE W(QUOTE A)))
(SETQ NONCON(REPLACE W(QUOTE B)))
(SETQ L PROBL)
LOOP3(COND((NULL L)(GO PROGRESS))
((M2(CADDAR L)NONCON NIL)(GO A)))
(SETQ  L(CDR L))
(GO LOOP3)
A(SETQ M PROBL)
(SETQ W CONST)
A2(SETQ W(SIMPTIMES(LIST W (CADAR M))))
(COND((EQ M L)(GO A1)))
(SETQ M(CDR M))
(GO A2)
A1(COND((M2 W(CADAR L)NIL)(GO KILL)))


(RPLACA(CDAR L)
  (SIMPQUOTIENT(LIST(CADAR  L)(SIMPDIFFERENCE(LIST(CADAR L)W)))))


(SETQ ANSW 0)
SKIP2(SETQ L PROBL)
LOOP4(COND((NULL L)(RETURN ANSW)))
(SETQ ANSW(SIMPTIMES(LIST(CADAR L)
```

```
  (SIMPPLUS(LIST(SIMPTIMES(LIST(CADDR(CDDDAR L))(CAR(CDDDAR L))))ANSW)))))
(SETQ L(CDR L))
(GO LOOP4)
FINISHED(SETQ ANSW(INTEGRATE B' VAR))
(GO SKIP2)

PROGRESS(COND((RAT8 B')(GO FINISHED))
(ONEMORE(RETURN(QUOTE(NO PROGRESS))))
((EQUAL POSEXPT 1)(SETQ ONEMORE T))
( (EQUAL ARCLOG 1)(SETQ ONEMORE T)))
(COND((NOT NINTXP)(GO LOOP)))
(SETQ W(M2 B'(QUOTE(TIMES(EXPT(A EQUAL(REPLACE NINTXP(QUOTE A)))
  (M TRUE1))(COEFFTT(D TRUE))))NIL))
(COND((NULL W)(ERROR(QUOTE NINTXP))))
(SETQ M(SIMPDIFFERENCE(LIST(REPLACE NINTXP(QUOTE N))
  (REPLACE W(QUOTE M)))))
(COND((NOT(NUMBERP M))(ERROR(LIST(QUOTE NINTXP)M)))
((ZEROP M)(GO LOOP))
((GREATERP M 0)(GO N1))
(ONEMORE(RETURN(QUOTE(NO PROGRESS NINTXP)))))
(SETQ ONEMORE T)
(GO LOOP)
N1(SETQ ANSW(LIST(QUOTE INTEGRAL) NIL (LIST(QUOTE QUOTE)B') (LIST(QUOTE QUOTE )
  VAR)))
(GO SKIP2)

KILL1 (SETQ PROBL(CDR PROBL))
KILL2(COND((NULL PROBL)(GO MAYBEONEMORE)))
(SETQ L(CAR PROBL))
(COND((CAR(CDDDDR(CDDDDR L)))(GO POSEXPT)))
(COND((NOT(CADDDR (CDDDDR L)))(GO KILL1))
((EQ(CAR(CADDDR L))(QUOTE LOG))(GO KILL1))
)
(SETQ FF(CADDDR(CDR L)))
(SETQ B'(CAR L))
(SETQ CONST(CADR L))
(SETQ NONCON(CADDR L))
(SETQ AORA' T)
(SETQ G(COND((EQ(CAR(CADDDR L))(QUOTE ARCSIN))
  (LIST(QUOTE EXPT)(CADR FF)(SIMPPLUS(LIST(CADDR FF)1))))
  (T(LIST(QUOTE LOG)(QUOTE E)(CADR FF)))))
(SETQ PROBL(CDR PROBL))
(SETQ ONEMORE NIL)
(GO GOGO)
KILL(SETQ OLDPROBL PROBL)
(GO KILL2)

MAYBEONEMORE(COND(ONEMORE(RETURN(QUOTE(I GIVEUP)))))
(PRINT(LIST(QUOTE ONEMORE)OLDPROBL))
(SETQ PROBL OLDPROBL)
(SETQ ONEMORE T)
(GO LOOP)

POSEXPT(COND((EQUAL(CAR(CDDDDR(CDDDDR L)))1)(GO KILL1)))
(SETQ FF(CADDDR(CDR L)))
(SETQ POSEXPT 1)
(SETQ AORA' T)
(SETQ B' (CAR L))
(PRINT(LIST(QUOTE POSEXPT)B'))
(SETQ CONST(CADR L))
(SETQ NONCON(CADDR L))
```

```
(SETQ G FF)
(SETQ PROBL(CDR PROBL))
(GO GOGO)
)))))

DEFINE((
(TRIGSUBST(LAMBDA(EXP)
(COND
((ATOM EXP)EXP)
((NOT(MEMBER(CAR EXP)(QUOTE(TAN COT SEC CSC))))
  (SIMP(MAPLIST EXP(FUNCTION(LAMBDA(C)(TRIGSUBST(CAR C)))))))
((EQ(CAR EXP)(QUOTE TAN))(SIMPQUOTIENT(LIST(LIST(QUOTE SIN)(CADR EXP))
  (LIST(QUOTE COS)(CADR EXP)))))
((EQ(CAR EXP)(QUOTE COT))(SIMPQUOTIENT(LIST(LIST(QUOTE COS)(CADR EXP))
   (LIST(QUOTE SIN)(CADR EXP)))))
((EQ(CAR EXP)(QUOTE SEC))(SIMPQUOTIENT(LIST 1(LIST(QUOTE COS)(CADR EXP)))))
)))))
```

BIBLIOGRAPHY

1. Anderson, R., "Syntax-Directed Recognition of Hand-Printed Two-Dimensional Mathematics," Memorandum 64, Project TACT, Harvard University, Cambridge, Mass., July 1967.

2. Baylor, G.W., and Simon, H.A., "A Chess Mating Combination Program," Proceedings 1966 Spring Joint Computer Conference, Spartan Books, Washington, D.C., pp. 431-447.

3. Black, F., "A Deductive Question Answering System," doctoral dissertation, Harvard University, Cambridge, Mass., 1964.

4. Bobrow, D.G., "A Question Answering System for High School Algebra Word Problems," Proceedings of the 1964 Fall Joint Computer Conference, Spartan Press, Baltimore, Maryland.

5. Bond, E., et.al., "An Experimental Formula Manipulation Compiler," Proceedings 1964 ACM National Conference, pp. K2.I-1-K2.I-11.

6. Brown, W.S., Hyde, J.P., and Tague, B.A., "The ALPAK System for Non-numerical Algebra on a Digital Computer - II," Bell System Tech. Journal XLIII, No. 2, 1964, pp. 785-804.

7. Brown, W.S., Rational Exponential Expressions and a Conjecture Concerning π and e, Bell Telephone Laboratories, Murray Hill, New Jersey, 1967.

8. Caracciolo di Forino, A., Spanedda, L. and Workenstein, N., "PANON-1B --A Programming Language for Symbol Manipulation," University of Pisa, Italy, 1966.

9. Caviness, B.F., "On Canonical Forms and Simplifications," doctoral dissertation, Carnegie Institute of Technology, 1967.

10. Christensen, C., "On the Implementation of AMBIT, A Language for Symbol Manipulation," Communications of the ACM, Vol. 9, No. 8, August 1966.

11. Crisman, P.A. (ed.), The Compatible Time-Sharing System: A Programmer's Guide (second edition), MIT Press, Cambridge, Mass., 1965.

12. Collins, G.E., "PM, A System for Polynomial Manipulations, " Communications of the ACM, Vol. 9, No. 8, August 1966, pp. 578-589.

13. Collins, G.E., "Subresultants and Reduced Polynomial Remainder Sequences," Journal of the ACM, Vol. 14, No. 1, January 1967, pp. 128-142.

14. Davis, M., Putnam, H., and Robinson, J., "The Decision Problem for Exponential Diophantine Equations," Annals of Math., Vol. 74, 1961.

15. Engelman, C., "MATHLAB: A Program for On-Line Assistance in Symbolic Computations," Proceedings 1965 FJCC, Spartan Books, Washington, D.C.

16. Evans, J.W., Harary, F., and Lynn, M.S., "On the Computer Enumeration of Finite Topologies," Communications of the ACM, Vol. 10, No. 5, May 1967, pp. 295-297, 313.

17. Evans, T.G., "A Program for the Solution of a Class of Geometry-Analogy Intelligence-Test Questions," Report AFCRL-64-884, Air Force Cambridge Research Laboratories, Hanscom Field, Mass., 1964. (A paper based on this dissertation was presented at the 1964 Spring Joint Computer Conference.)

18. Feigenbaum, E.A., and Feldman, J. (eds.), Computers and Thought, McGraw-Hill, New York, 1963.

19. Fenichel, R.R., "An On-Line System for Algebraic Manipulations," doctoral dissertation, Harvard University, July 1966, (also available as Report MAC-TR-35, Project MAC, MIT, Cambridge, Mass., Dec. 1966).

20. Fenichel, R.R., and Moses, J., "A New Version of CTSS LISP," Memorandum MAC-M-296, Project MAC, MIT, Cambridge, Mass., Dec. 1966.

21. Goldberg, S.H., "Solution of an Electrical Network Using a Digital Computer," M.S. Thesis, MIT, Cambridge, Mass., 1959.

22. Greenblatt, R.D., Eastlake, P.E., and Crocker, S.D., "The Greenblatt Chess Program," to appear in the Proceedings of the 1967 Fall Joint Computer Conference.

23. Guzman, A., and McIntosh, H.V., "CONVERT," Communications of the ACM, Vol. 9, No. 8, August 1966, pp. 604-615.

24. Haan, Bierens de, Nouvelle Tables d'Intégrals Definées, G.T. Stechert, New York, 1939.

25. Hardy, G.H., The Integration of Functions of a Single Variable, second ed., Cambridge Univ. Press, Cambridge, England, 1916.

26. Henneman, W., private communication, 1966.

27. Ince, E.L., Integration of Ordinary Differential Equations, 7th ed., Oliver and Boyd, London, 1963.

28. Iturriaga, R., "Contributions to Mechanical Mathematics," doctoral dissertation, Carnegie Institute of Technology, Pittsburgh, Penna., April, 1967.

29. Jolley, L.B.W., Summation of Series, second edition, Dover, New York, 1961.

30. Kamke, E., Differentialgleichungen, Lösungsmethoden and Lösungen, Vol. I, second edition, J.W. Edwards, Ann Arbor, Mich., 1945.

31. Kaplansky, I., An Introduction to Differential Algebra, Paris, 1957.

32. Klerer, M., and May, J., "An Experiment in a User Oriented Computer System," Comm. A.C.M., vol. 7, No. 5, 1964, pp. 290-294.

33. Korsvold, K., "An On-Line Algebraic Simplify Program," Art. Intell. Project Memo 37, Stanford Univ., Palo Alto, Calif., 1966.

34. McCarthy, J., et al., LISP 1.5 Programmer's Manual, MIT Press, Cambridge, Mass., 1963.

35. McIntosh, H.V., "Degeneracy of the Magnetic Monopole," Bull. of Amer. Physical Society, series II, vol. 12, p. 699, 1967.

36. Manove, M., Bloom, S., and Engelman, C., "Rational Functions in MATHLAB," Proc. of the I.F.I.P. Working Conf. on Symbol Manipulation Languages, Pisa, Italy, Sept. 1966 (to appear).

37. Martin, W.A., "Symbolic Mathematical Laboratory," doctoral dissertation, MIT, Cambridge, Mass., Jan. 1967 (also Report TR-36, Project MAC, MIT).

38. Maurer, W.D., "A Table of Integrals Involving the Error Function and Related Functions," Argonne National Laboratory, Reactor Engineering Group, 1958.

39. Maurer, W.D., "Computer Experiments in Finite Algebra," Comm. ACM, vol. 9, No. 8, August 1966, pp. 589-603.

40. Millen, J.K., "CHARYBDIS: A LISP Program to Display Mathematical Expressions on Typewriter-Like Devices," presented at ACM Symposium on Interactive Systems for Experimental Applied Mathematics, Wash., D.C., August 1967.

41. Minsky, M.L., "Steps Toward Artificial Intelligence," in Computers and Thought, McGraw-Hill, New York, 1963.

42. Moses, J., "Solution of Systems of Polynomial Equations by Elimination," Comm. of the ACM, Vol. 9, No. 8, August 1966, pp. 634-637.

43. Newell, A., Shaw, J.C. and Simon, H.A., "Report on a General Problem Solving Program," in Computers and Thought, McGraw-Hill, New York, 1963.

44. Newell, A., Shaw, J.C. and Simon, H.A., "Empirical Explorations of the Logic Theory Machine: A Case Study in Heuristics," in Computers and Thought, McGraw-Hill, New York, 1963.

45. Newell, A., and Ernst, G., "The Search for Generality," Proc. IFIP Congress, 1965, Vol. I, pp. 17-24.

46. Newell, A., "Some Problems of Basic Organization in Problem Solving Programs," in Self-Organizing Systems 1962, Yovits, M., Jacobi, G. and Goldstein, G., editors, Spartan Books, 1962, pp. 393-423.

47. Norton, L.M., "ADEPT - A Heuristic Program for Proving Theorems of Group Theory," Tech. Report TR-33. Project MAC, MIT, Cambridge, Mass., Oct. 1966.

48. Perlis, A.J., Iturriage, R., and Standish, T.A., "A Definition of Formula ALGOL," Depart. of Computer Science, Carnegie Inst. of Tech., Pittsburgh, Penna., March 1966.

49. Persson, A., "Some Sequence Extrapolating Programs: A Study of Representation and Modeling in Inquiring Systems," Tech. Report CS50, Computer Science Depart., Stanford Univ., Palo Alto, Calif., Sept. 1966.

50. Peterson, F.P., and Sims, C., "The Formulation of the Statement of a Cobordism Structure Theorem," Mathematical Algorithms, Vol. I, No. 3, July 1966, pp. 1-11.

51. Petit Bois, G., Tables of Indefinite Integrals, Dover Publications, New York, 1961.

52. Richardson, D., doctoral dissertation, Univ. of Bristol, Bristol, England, 1966.

53. Risch, R.H., "The Problem of Integration in Finite Terms," Report

SP-2801, Systems Development Corp., Santa Monica, Calif., March 1967.

54. Ritt, J.F., Integration in Finite Terms, Columbia Univ. Press, New York, 1947.

55. Sammet, J.E., "Survey of Formula Manipulation," Comm of the ACM, Vol. 9, No. 8, Aug. 1966, pp. 555-569.

56. Sammet, J.E., "An Annotated Description Based Bibliography on the Use of Computers for Non-Numerical Mathematics," Computing Review, Vol. 7, No. 4, July 1966, pp. B1-B31.

57. Segovia, R. and McIntosh, H.V., "Computer Analysis of Finite Groups," presented at 1966 Fall Joint Computer Conf.

58. Slagle, J.R., "A Heuristic Program that Solves Symbolic Integration Problems in Freshman Calculus, Symbolic Automatic Integrator (SAINT)," doctoral dissertation, MIT, 1961 (a paper based on this thesis is in Computers and Thought, McGraw-Hill, New York, 1963).

59. Slagle, J.R., "Experiments with a Deductive Question Answering Program," Comm. ACM, Vol. 8, 1965, pp. 792-798.

60. Spiegel, M.R., Applied Differential Equations, Prentice-Hall, Englewood Cliffs, New Jersey, 1958.

61. Tarski, A., A Decision Method for Elementary Algebra and Geometry, second ed., Univ. of Calif. Press, Berkeley, Calif., 1951.

62. Teitelman, W., "FLIP - A Format List Processor," Memo MAC-M-263, Project MAC, MIT, Cambridge, Mass., 1965.

63. Tobey, R.G., Bobrow, R.J. and Zilles, S., "Automatic Simplification in FORMAC," Proc. 1965 FJCC, Spartan Books, pp. 37-57.

64. Tobey, R.G., doctoral dissertation, Harvard Univ., Cambridge, Mass., 1967.

65. van der Waerden, B.L., Modern Algebra, vol. 1, Frederick Unger, New York, 1953.

66. Weizenbaum, J., "ELIZA - A Computer Program for the Study of Natural Language Communication between Man and Machine," Comm. ACM, Vol. 9, No. 1, Jan. 1966, pp. 36-45.

BIOGRAPHY OF THE AUTHOR

Joel Moses was born in [redacted], on [redacted]. He entered the United Stated on September 1, 1954 and became a naturalized citizen in 1960. After graduating from Midwood High School, Brooklyn, New York, in June 1959, he entered Columbia College from which he graduated Magna Cum Laude in June 1962. During this time he held a New York State Engineering and Science Scholarship. He then enrolled in the Graduate Faculties of Columbia University and held an IBM Fellowship for the year 1962-1963. He received a Master of Arts in Applied Mathematics in June 1963. Since that time he has been a graduate student at MIT in its Department of Mathematics and a research assistant at its Research Laboratory of Electronics and Project MAC.

The author has been employed by IBM's Watson Research Laboratory, New York, New York, and Boston Advanced Programming Department, Cambridge, Massachusetts, and by the Lincoln Laboratory, Lexington, Massachusetts. He has accepted employment as Assistant Professor in the Department of Electrical Engineering at MIT.

His publications include:

"Solution of Systems of Polynomial Equations by Elimination," Communications of the ACM 9, 8 (August 1966), pp. 634-637.

The author is a member of The American Mathematical Society, The Association for Computing Machinery, Phi Beta Kappa, and Sigma Xi.

UNCLASSIFIED

Security Classification

DOCUMENT CONTROL DATA - R&D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)* Massachusetts Institute of Technology Project MAC	2a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED 2b. GROUP None

3. REPORT TITLE

Symbolic Integration

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*

Ph.D. Thesis, Department of Mathematics, September 1967

5. AUTHOR(S) *(Last name, first name, initial)*

Moses, Joel

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
December 1967	268	66

8a. CONTRACT OR GRANT NO. Office of Naval Research, Nonr-4102(01) b. PROJECT NO. NR 048-189 c. RR 003-09-01 d.	9a. ORIGINATOR'S REPORT NUMBER(S) MAC-TR-47 (THESIS) 9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*

10. AVAILABILITY/LIMITATION NOTICES

Distribution of this document is unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
None	Advanced Research Projects Agency 3D-200 Pentagon Washington, D. C. 20301

13. ABSTRACT

SIN and SOLDIER are heuristic programs written in LISP which solve symbolic integration problems. SIN (Symbolic INtegrator) solves indefinite integration problems at the difficulty approaching those in the larger integral tables. SIN contains several more methods than are used in the previous symbolic integration program SAINT, and solves most of the problems attempted by SAINT in less than one second. SOLDIER (SOLution of Ordinary DIfferential Equations Routine) solves first-order, first-degree, ordinary differential equations at the level of a good college sophomore and at an average of about five seconds per problem attempted. The differences in philosopy and operation between SAINT and SIN are described, and suggestions are made for extending this work.

14. KEY WORDS

Algebraic manipulation	Multiple-access computers	Symbolic integration
Computers	On-line computers	Time-sharing
Machine-aided cognition	Real-time computers	Time-shared computers

DD FORM 1 NOV 65 1473 (M.I.T.)

UNCLASSIFIED

Security Classification