IDEA
-A Symbolic Integration Program-

by

Leon Welliver, B. A.

1976

ABSTRACT

A program which solves symbolic differentiation and integration problems is described. The program is written in a language which makes heavy use of recursion, and so recursive techniques are used widely. The differentiation program solves problems containing all the common types of functions found in basic calculus primers, and the integration routine, while not being quite as complete, nevertheless solves a wide variety of problems and is a useful tool. The view is taken that simplification and integration routines should as far as possible preserve the structure of the expression inputted to them, and parallels between the goals of integration and of simplification are drawn. A discussion of possible improvements and of further research is made.

D E C L A R A T I O N

This thesis is presented to the University of St. Andrews in fulfilment of the requirements for the degree of Master of Science. It embodies the results of research at the University. It has not been accepted in the fulfilment of the requirements of any other degree or professional qualification. It has been composed wholly by myself.

Leon Welliver

TABLE OF CONTENTS

A C K N O W L E D G E M E N T

The usual sort of gratitude is hereby expressed to my advisor, Mr. D. Turner, who has taught me the utopia of a Fortranless society as well as the hypocrisy of Tory government, and to my parents, Mr. and Mrs. C. L. Welliver of Los Angeles who financed this whole adventure.

CHAPTER I
BACKGROUND

1.1 Introduction

IDEA is an acronym for Integration and Differentiation Evaluation Algorithm. It seeks a differentiation or an anti-differentiation, as specified by the user, of a given elementary function. Elementary function is defined recursively, and Backus-Normal Form facilitates its definition:

```
⟨variable⟩ ::=  sequence of letters
⟨constant⟩ :: =  ⟨variable⟩|⟨integer⟩
⟨elem-func⟩ :: =  ⟨constant⟩|⟨neg-op⟩⟨elem-func⟩|
                  ⟨elem-func⟩⟨arith-op⟩⟨elem-func⟩|
                  ⟨log-op⟩⟨elem-func⟩|
                  ⟨trig-op⟩⟨elem-func⟩|
                  ⟨hyper-op⟩⟨elem-func⟩
⟨neg-op⟩ ::  -
⟨arith-op⟩ :: =  + | -  | / |* | ** (exponentiation)
⟨log-op⟩:: =  expn(e to the power of) | ln(napierian log) |
              log(log to any other base)
⟨trig-op⟩:: =  sin |cos |tan |cot |sec |csc
               arcsin |arccos |arctan |arccot |arcsec |arccsc
⟨hyper-op⟩:: =  sinh |cosh |tanh |coth |sech |csch|
                arcsinh |arccosh |arctanh |arcsech |arccoth |arccsch
```

The hyperbolic functions are not handled by the integration routine, but they are handled by the differentiation routine.

It is seen that the elementary functions above are quite like the set of functions considered in most first primers on the calculus. The goal of IDEA is to find the anti-differentiation or differentiation, as specified by the user, of elementary functions in terms of other elementary functions.

Centuries have passed since Newton and Leibnitz discovered the calculus simultaneously. During that time, the basic principles of calculus have been well developed, so that today the methods used in solving calculus problems are well known and very systematized. The experience of years allows virtually all methematicians to tackle a given calculus problem in often

the same way. While there is no "perfect solution" to calculus problems, the hints at solution within the problem usually suggest similar methods to different mathematicians. The schoolboy learning the calculus might stew all night over an "impossible integration" only to have a tutor or his lecturer the next morning non-chalantly say, "This is the old thus-and-such a case" and immediately supply the magic substitution. The student is perhaps chagrined, but the next time he comes across such a problem, he will know what to do. The famous mathematician Geoffrey Hardy expressed similar sentiment in writing at the beginning of a calculus monograph:

> This pamphlet is intended to be read as a supplement to the accounts of 'Indefinite Integration' given in text-books on Integral Calculus. The student who is only familiar with the latter is apt to be under the impression that the process of integration is essentially 'tentative' in character, and that its performance depends on a large number of disconnected though ingenious devices. My object has been to do what I can to show that this impression is mistaken, by showing that the solution of any elementary problem of integration may be sought in a perfectly definite and systematic way.[10]

If much of the integration problem can be met by recognising the "catch" in the integrand, then it should be possible to program a computer to recognise the catch and supply a solution. This sort of ide a has been with man far longer than his computers; the basic patterns of integrals are written down along with advice for their solutions in instruments now called integral tables.

IDEA, though, is not strictly an integral table look-up program. It searches for clues in a given integrand and heuristically supplies suggestions for the solution of the integral based on the clues found. This avoids the need to completely identify an integrand in a look-up table as

a given method of integration completely solves very different looking integrals, e.g. change-of-variable substitution solves the algebraic

$$\int \frac{x\,dx}{\sqrt{x^2 - 4}}$$

as well as the trigonometric

$$\int \sin x \cos x \; dx$$

By using a certain resourcefullness in finding a method of integration, the machine exhibits artificially intelligent behaviour. Actually, it is the programmer who supplies the machine his experience as to what works best in a given situation. IDEA is presented with a theme of experience and intelligence.

1.2 Purpose

The purpose of IDEA is to exhibit a modern symbolic integration package along with suggestions on how it might be extended. By "modern" it is meant an implementation using techniques of integration currently in use and using the most high-level programming techniques. While the program documented here is not as complete an integrator as some others that were written, it does exhibit some novel approaches to the problem as well as some good thoughts towards its extension in the final analysis. It is intended to describe a program which solves a wide variety of non-trivial integrations.

1.3 The approach to the problem - artificial intelligence

In section 1.2, a specific problem was defined which, if solved by humans, would be commonly thought to require intelligence on the part of the human solver. In programming a computer to solve the same problem,

the program becomes an artificial intelligence program. The program does not churn through some routine which always leads to an answer as it would in, say, a program to find the factorial of a number. The machine uses some "rules-of-thumb" supplied by the programmer in hopes of finding a solution which may not even exist.

It was Turing that first wrote about the possibilities of machines performing "intelligent" tasks.[29] He realised the difficulties in getting a machine to exhibit an independent approach to novel situations, i.e. to modify its behaviour in adjustment to a given new situation. Machine intelligence is a simulation of human intelligence, he stated, and this definition has changed little to this day.

Many approaches to artificial intelligence have been proposed since Turing's paper. One approach which finds wide application to very many artificial intelligence problems is a "heuristic graph search, or state-space", approach. This approach has direct applicability to the problem of symbolic integration and is chosen to mobilise IDEA. Before this applicability is demonstrated, some of the important literature on graph traversal are surveyed.

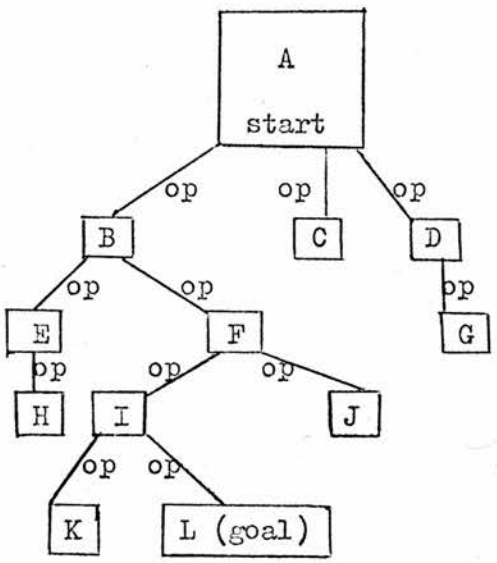
Fogel, Owens, and Walsh wrote a pioneering book on the procedures involved in writing a program which exhibits artificial intelligence.[8] They describe the problem as an evolution of a primary state to a goal state. Doran and Michie wrote good papers on the representation of intelligence problems by graph and how to traverse the graph to seek a solution to the original problem.[4,18] These authors are among the first to discuss heuristic search. Michie and Ross improved upon this work by writing an adaptive graph traverser which modified dynamically its problem transformation operators and improved on the heuristic evaluation functions.[17]

Heuristic search using graphs is further explored by Pohl.[25] In another paper, Pohl concerned himself with the effect of error in heuristic search, i.e. just how far off course would a program go by taking a given wrong turn.[24] Amarel gives good suggestions on decision making during heuristic procedures in one of his papers.[1] Comprehensive summaries of artificial intelligence procedures are given in texts by Slagle, Nilsson, and Hunt.[28,23,13]

The knowledge from all of the above papers and books, together especially with the comprehensive pictures provided by Slagle, Nilsson, and Hunt, go together to form the basic artificial intelligence strategy for IDEA. These authors place emphasis on representation of problems in artificial intelligence by a graph. More specifically, the graph is usually a directed graph or a tree. The top of the tree (root node) is the initial state given in the problem. A goal is sought from this starting point. One example of a goal for artificial intelligence would be a deduction of some sort. This means that the root node is the "given" part of the proof, and the goal is the "to prove" part. In IDEA, the root node would be an integrand, and the goal node would be its anti-differentiation.

The crux of the problem lies in getting from the start node to the goal node. A human faced with such a task would proceed stepwise towards his goal, e.g. he would present sub-propositions towards proving a theorem, or he would transform an integrand possibly several times before creating a form he could find in his integral tables. In the graphical analysis, the forms intermediate to the start node and the goal node are the steps in the solution to the problem. Each step is deduced from some other step, or from the start node, and the logic which allows a step to be taken is called an operator. Each step is represented graphically as a node on the

tree. Nodes lower down the tree are deduced from nodes higher up the tree. The line segments between nodes are the operators. For a symbolic integration problem, the operators might have names like "change-of-variable substitution" or "trigonometric substitution." If the goal node is ever reached from a node higher up the tree (via some operator) which in turn is deduced through some number of steps from the start node, then the path from the start node to the goal node is said to be a solution. The situation is pictured in figure I.1.



A solution to problem A would be ABFIL

Figure I.1

In creating a heuristic search tree, different strategies may be followed. A breadth-first search method expands its nodes to all their succesors and the next level down before proceeding to the following level. In figure I.1, the nodes would be created in the order ABCDEFGHIJKL. Since all the successors are tried, a breadth-first method must necessarily find a solution to a given problem since all the nodes are exhaustively searched. The pitfall in this method is that if a node has a great many

successors then generating them all could be a very expensive process.

A depth-first method is one in which a node is expanded to its first successor and that node is then expanded to its first successor and so on until a node can no longer be expanded or a solution is found. If the tree in figure I-1 had been created by depth-first search, then its nodes would have been created in the order ABEHFIKL by which time a solution is found. The danger here is that a node may have infinite depth without leading to solution. Safeguards should be added to ensure that the search routine does not runaway on a wild goose chase. One improvement to a depth-first routine would be to provide a node evaluation function which would decide which node is closest to solution and therefore should be expanded first.

IDEA uses a special case of depth-first search. All the immediate successors, i.e. all the nodes one level down from the start node are created first. The next step is to expand the first node generated in this way to all of its possible successors and so on. The logic of this is explained in Chapter V. Again using figure I-1 as an example, the IDEA strategy would generate the nodes of that tree in the order ABCDEFHI JKL by which time a solution would have been found so further generation of nodes stops.

1.4 Previous work

Two major symbolic integrators have been implemented. One is SAINT by Slagle and the other is SIN by Moses.[27,22] They both use LISP as a language for writing their strategies down for the computer.[16] They are briefly described here.

I.4a SAINT

SAINT is one of the early pioneering programs in artificial intellig-

ence. It was designed to show that a machine can manifest intelligent problem-solving behaviour. The original integrand is placed on a goal list, and to this goal list may be added other integrands the solutions of which might lead to the solution of the original integrand. Slagle judged that the methods of integration factorization of constant, the integral of a sum is the sum of the integral, and change-of-variable substitution are always, or nearly always, appropriate. If these substitutions are inadequate to solve the problem, then the problem is transferred to a "temporary goal list." There the difficulty of the problem is measured and sub-problems which were on the temporary goal list are transferred in order of increasing difficulty to a "heuristic goal list" which applied transformations to the integrand. Not included in the program was a rational function integration procedure. Whenever multiple integration or definite integration was a trivial extension single-variable indefinite integration, this also could be done by SAINT. Slagle claims that SAINT handles calculus problems "at the level of a good college freshman" and took a first-year calculus examination with good proficiency.

1.4b SIN

A more complete symbolic integration program than SAINT written by Moses is called SIN. SIN relies on a much tighter analysis of the problem domain, i.e. integration, than does SAINT in order to rely less on tree search and to obtain a more straightforward solution. Whereas SAINT is likely to try several transformations on a given integrand and then decide which of the resulting integrands is best for continuation, SIN scrutinizes the integrand very closely in hopes of seeing the best transformation immediately.

SIN is divided into three stages. The first stage tries to solve the integration problem by some cheap and quick-to-use method such as change-

of-variable substitution. If the integral requires a more involved solution, a routine called "form" seeks out an appropriate transformation. Altogether, there are eleven transformations that might be tried. The reader is referred to the original publications for a description of these methods. If these eleven methods fail, then the integrand is passed on to a third stage which applies general methods of integration. A great deal of analysis and resources are used by these general methods; they are expensive to use.

The SIN integration system is used by another program, SOLDIER, in order to solve ordinary differential equations. Special routines are also provided to solve definite integration problems, even those which cannot be evaluated by substituting limits for the variable of integration, e.g.

$$\int_0^{2\pi} r(\sin x \cos x)dx$$

where r is a rational function. (The solution involves substitution of complex exponentials for the trigonometric functions and evaluating on a unit-circle using the residues of the complex functions)

1.5 Choice of programming language

IDEA is written in a language called SASL.[30] SASL contains tools for handling lists (and trees represented by lists) and is concise. Its structures are more elegant than those of other list processing languages, e.g. LISP, and thus becomes easier to use than these other languages. SASL makes heavy use of recursive techniques, and thus is particularly well-suited to operations involving recursively defined data structures, e.g. trees and lists, and recursively defined programming operations, e.g. application of the chain rule in differentiation.

Of especial significance to the integration problem at hand is SASL's

approach to functions, which is different to many of the languages in common use today. The definition of a function, represented by that function's name, may be passed as an item in a list and then evaluated later by plugging appropriate parameters into it. Thus, if each integration method is written as a function, a list of methods applicable to a given integrand may easily be represented. The use of heuristic evaluation functions, which in IDEA are implicit, facilitates the correct choice of method for a given integrand.

For the reader's convenience, the BNF of SASL is given in appendix B. Readers are referred to Turner for details.

CHAPTER II
INPUT/OUTPUT

2.1 The basic idea

The input/output scheme of IDEA balances the needs of user and of machine. The user will want to use Fortran-Algol like infix notations to represent his expressions because that is what is natural to him and easiest for him to understand. The machine finds greatest utility in forms which have no ambiguity as to their meaning and which can be easily examined for the integration and differentiation operations. The IDEA machine uses a tree-structured form of polish notation. It is a simple matter to keep the user and the internal representations separate.

2.2 The user interface

The user actually completes a SASL program by issuing a differentiate or integrate command. He merely inputs the word "differentiate" or "integrate" followed by two strings enclosed in SASL quotes. The first string is his infix representation of his expression. The second is the variable of differentiation or integration. If the user misspells the command, or misuses the SASL quotes, or in any other way violates the rules for forming SASL programs, he may generate SASL error messages which are usually quite clear as to the source of difficulty. If the infix expression itself is incorrectly formed, an error message of the form "syntax error: -error message-" will be generated by IDEA. If none of these catastrophes occur, a normal output is produced.

The output of a differentiation takes the form

"the derivative of"
-user's infix expression-
"with respect to" -variable of differentiation-"is"
-infix representation of answer-

The differentiator is demonstrated by the following reproductions of actually

inputted problems

```
differentiate 'sin x" 'x"
<compiled>
the derivative of
sin x
with respect to x is
cos x

differentiate 'y**3/3" 'y"
<compiled>
the derivative of
y**3/3
with respect to y is
y** 2

differentiate 'arctan(u**2)" 'u"
<compiled>
the derivative of
arctan(u**2)
with respect to us is
(2*u) / (1+u**4)

differentiate 'arctan(a*x**2+b)" 'x"
<compiled>
the derivative of
arctan (a*x**2+b)
with respect to x is
a*2*x/(1+(a*x**2+b)**2)

differentiate 't**n" 't"
<compiled>
the derivative of
t**n
with respect to t is
n*t**(n-1)

differentiate 't**n" 'n"
<compiled>
the derivative of
t**n
with respect to n is
ln t*t**n

differentiate 'arccsch(cosh x)" 'x"
<compiled>
the derivative of x
arccsch(cosh x)
with respect to x is
-(sinh x/cosh x*(1 +(cosh x)**2)**(1/2)
```

```
differentiate 'dog(x)" 'x"
<compiled>
the derivative of
dog(x)
with respect to x is
dog' x
```

Chapter III discusses the actual inner workings of the differentiation routine.

The output of an integrate command is similar to that for a differentiate command. If IDEA cannot do the integration, then it prints out, " i am stuck." If it can, it prints out in the form:

"the integral of"
-user's infix expression-
"in" -variable of integration- "is"
-infix representation of answer-

Consider the following examples of output:

```
integrate 'x**3" 'x"
<compiled>
the integral of
x**3
in x is
x**4/4

integrate '(sin x)*(cos x)" 'x"
<compiled>
the integral of
(sin x)*(cos x)
in x is
-((cos x)**2/2)

integrate '(tan theta)**5" 'theta"
<compiled>
the integral of
(tan theta)**5
in theta is
(2*(tan theta)**4-4*((tan theta)**2-2*ln(sec theta)))/8
```

The integration routine is discussed in Chapter V.

2.3 The construction of an input string

The rules for construction of strings to give IDEA are kept as simple as possible. The algebraic operators are represented in the usual Fortran-Algol way: + for addition, - for subtraction, * for multiplication, / for division, and ** for exponentiation. Roots are represented by fractional exponents.

Only integers are allowed as numbers. This is due in part to the lack of real numbers in SASL and also because it is usual to represent all numbers rationally in the calculus.

Variable names may be any combination of the 26 letters of the alphabet. This allows a very wide variety of variable names and the beginnings and ends of variable names are easily detected by SASL test operators.

Functions are a variable name followed by a space and then another variable name or an integer as an argument, or followed by a legal expression in parentheses. IDEA uses standard representations for most of the functions commonly considered in the calculus. These mnemonics are shown in table II-1 and are easy to recognise. "Expn" is a function which raises the napierian base e to the power of its argument. "Ln" the napierian logarithm function. "Log" is the logarithm to any other base.

trig functions	inverse trig functions	hyperbolic functions	inverse hyperbolic functions	general
sin	arcsin	sinh	arcsinh	expn
cos	arccos	cosh	arccosh	ln
tan	arctan	tanh	arctanh	log
cot	arccot	coth	arccoth	
sec	arcsec	sech	arcsech	
csc	arccsc	csch	arccsch	

TABLE II-1

The negative is represented, as usual, by a minus sign. Internally, this will be represented as an underscore ("_") so that it may be easily distinguished from subtractions.

The user uses all these symbols in combination with parentheses to group subexpressions if he wishes. All these follow the normal rules for formation of expressions by algebraists.

Examples of legal IDEA expressions:

```
a+b+c
a*b+c
cos x
sin(x)
-(a+b)
(a+b)/(c+d)
x**2+7
function (x**3+b)*a
```

Note that all variable names which have no argument following them are assumed to be variables and not functions.

2.4 Internal representation

Infix representation has problems in definition of operator precedence and handling of parentheses which are alleviated by translation of the input string into polish notation. Each polish string has a corresponding tree structure equivalent to it. Since so many of the tasks of differentiation, integration, and simplification are defined recursively, since SASL makes heavy use of recursion, and since trees are defined recursively and are easily operated on in recursive fashion, trees are the chosen data representation for IDEA. Trees are simulated by SASL lists. Every arithmetic operator, plus, minus, multiply, divide, and exponentiate is a SASL 3-list consisting of the operator followed by its two arguments. The negative sign and all the other functions which are in IDEA are represented by 2-lists, the negative sign (represented by the underscore"_"), or the name of the function, followed by its argument. Thus all operations are repres-

ented naturally.

The routine for creating a tree expression from an infix string is the IDEA routine "exp." This routine utilises a standard type recursive-decent algorithm such as that described by Gries.[9] This method is easy to implement, follows the BNF of the grammar well, and neatly generates error messages when needed. This is a "top-down" technique, and associates operators to the right. This is natural for exponentiation and for stacking function calls.

Examples

a**b**c = a**(b**c)
sin cos x = sin(cos(x))

The view is taken here that right association is also most natural for divisions and multiplications.

Example

$$a/b*c = \frac{a}{b*c}$$

It is not natural for additions and subtractions. If it were, then an expression as a+b-c+d would be considered as a+b-(c+d) and the meaning of the plus sign in fron of "d" would be reversed. An extra level of recursion in the subroutine "terms" avoids this anomaly.

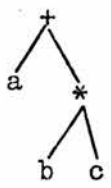
The BNF of the grammar is given here:

⟨exp⟩ ::= ⟨exp⟩⟨add-op⟩⟨term⟩ | ⟨term⟩
⟨term⟩ :: = ⟨factor⟩⟨mult-op⟩⟨term⟩ | ⟨factor⟩
⟨factor⟩ :: = ⟨arg⟩**⟨factor⟩ | ⟨arg⟩
⟨arg⟩ :: = (⟨exp⟩) / -⟨arg⟩ | ⟨number⟩ | ⟨fnexp⟩
⟨number⟩ :: = ⟨digit⟩⟨number⟩ | ⟨digit⟩
⟨fnexp⟩ :: = ⟨name⟩ | ⟨name⟩⟨arg⟩
⟨name⟩ :: = ⟨letter⟩⟨name⟩ | ⟨letter⟩
⟨letter⟩ :: = ⟨any letter of the alphabet⟩
⟨digit⟩ :: = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
⟨add-op⟩ :: = + | -
⟨mult-op⟩ :: = * | /

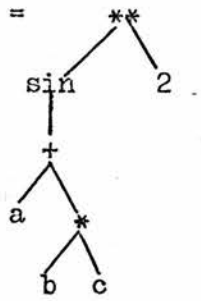
Examples of tree structures

a+b =

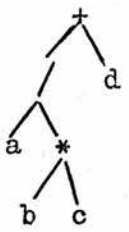
a+b*c =



sin(a+b*c)**2 =



a/b*c+d =



Monadic operators are distinguished from diadec operators dynamically by merely counting the successors of the operator node.

2.5 Output

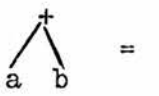
These tree structures may be outputted in infix by calling the procedure "output". The method used incorporates some of the ideas of Weissman.[31] The method is short and very easy to implement. In this routine as well as all the rest of the routines, the utmost care is taken that the outputted form resembles the inputted form so that the user may easily recognise it.

CHAPTER III
DIFFERENTIATION

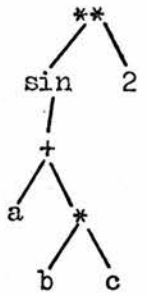
Any integration routine must have a differentiation routine as a trivial part of the program. In IDEA, the differentiation routine may be called on its own if all that is desired is the differentiation of an expression. IDEA uses a functional approach to the problem. This approach is sugared by SASL.

By the time an expression is passed to the differentiation routine, it has had a tree structure representation as described in Chapter II made for it. The tree structure is represented by a SASL list.

Examples



= %+,a,b,



= '**",('sin",(%+,a,(%*,b,c,),),),),2,

Most of the work of differentiation is done by the parser because the appropriate rule of differentiation is implied immediately by merely examining the first operator in the list. In the first example above, the first operator is plus so the rule is to sum the derivatives of the two arms of the addition. This is done by recursive calls to the differentiation function "diff." In a similar way, the second example above is examined and deemed to require a rule of differentiation for exponentiations.

The differentiation algorithm is then easy to describe. If the object to be differentiated is the same as the variable of differentiation, then the answer is one. If it is any other single entity, i.e. if it is atomic,

then the answer is zero. Otherwise, the head operator is passed to the function "diffop" whose value is a function which operates on the arms or arm of the operator according to the rules of differentiation. The chain rule is applied by recursive calls to "diff." In this way, one routine handles the differentiation rules for both the monadic and diadic functions since the routine "diffop" returns a function peculiar to the particular operator being examined.

Examples taken from actual coding (see appendix A)

```
s=%* -> Lambda (left, right,).
     %+,(%,*,left,diff right var,),(%*,right diff left var,),;

s='sin" -> lambda(operand,).
       %*,('cos",operand,),diff operand var,;
```

These examples hopefully clarify the working of the differentiator even to readers unfamiliar with the language SASL. The parameter s holds the leading operator of the list and the SASL reserved word lambda is an entity serving to identify functions in much the same way as lambda is used in Church's lambda calculus.[3] If the operand in the second example were x, and the variable of differentiation, held in var, were also x, then "diff" would return the tree structure

```
%*,('cos",'x",),1.
```

which would be simplified to

```
'cos", 'x",
```

by the simplifier described in Chapter IV.

The rules of differentiation are searched by linear search. This method is reasonably efficient for this task as the amount of rules to be searched is small (31). The rules are ordered for search so that the rules most likely to be appropriate, i.e. the arithmetic operator rules, are tried first. An expression not recognised by any of the rules is assumed by

default to be a monadic function. A prime (" ' ") is affixed to the name of the function and the chain rule applied.

Example

```
differentiate 'dog(x)" 'x"
⟨compiled⟩
the derivative of
 dog(x)
with respect to x is
dog' x
```

A complete list of the differentiation rules are appendix C.

CHAPTER IV
SIMPLIFICATION

4.1 The goal of simplification

Another important sub-program in an integration routine is an algebraic simplification algorithm. The many intermediate results generated during the course of an integration can have many terms in them which could be gathered together, divided down to unity, or some other such routine which is associated with simplification. Unless simplification can take place, the already difficult task of integration can become monstrously difficult if not impossible.

One great obstacle in the creation of a suitable simplification routine is the conflicting requirements set on the simplifier and another is the vague definition of the term "simplification." Moses in seeking to answer the questions above classifies the various approaches to the problem without ever supplying a definition of simplification in concrete terms.[20] The term "algebraic simplification" evokes different pictures in the minds of different people.

Certainly in context with the term simplification are certain rules such as addition to zero is an identity as is multiplication by unity. Beyond these basics it would seem that simplification needs a context, an environment, in which to work. The environment concerned here is integration. A "y" in an expression is a constant when integrating with respect to "x". the "y" takes on a very different meaning when integrating in "y." Since integration of the terms of a polynomial in the variable of integration is trivial, it might be argued that

$$\int (x+a)^n \, DX$$

should always be expanded by the simplifier and integrated termwise. Slagle takes this point of view in his SAINT program. But if the n in the above

expression is equal to 1,000 then expansion, although yielding a form which is trivial to integrate, is nevertheless very messy and, it is thought, much less pleasing to the eye than would be a form which made a substitution for x+a to give

$$\int (x+a)^{1000}\ dx = \frac{(x+a)^{1001}}{1001} + C$$

Also, IDEA will assume that, as far as is practicable, the user wishes to see the anti-differentiation of his expression take the same form as that which he used for input. It is recognised that a radical transformation, such as making trigonometric function into a complex exponential, might facilitate the integration by allowing the use of some high-brow mathematics, but the user will get his answer in a form which he might not recognise as having relevance to the form he inputted. In the above example, if the user wishes an answer which has the full 1002 terms in it, then he will put in the expression with the full 1001 terms in it. The simplifier used by IDEA has very few simplification rules incorporated into it and does its utmost to preserve the structure inputted to it. Integration is kept separate from simplification.

There is a subtle point in the preceding discourse which bears emphasis. Users generally present expressions to symbol manipulators in the form that they do because they have some reason for doing so. The terms of a polynomial are written down usually in ascending or descending powers of the independent variable because it allows readers to see easily pertinent characteristics of the polynomial, e.g. the degree, powers with zero coefficients, and the constant term. If a user writes down the terms of a polynomial in random order then it is proposed here

that he had some compelling reason for doing so. If the simplifier rearranges the inputted expression, then it may mask the context of input from the routine for which the simplifier works. A paraphrase of the golden rule as applied to routines which utilise simplifiers might go "give unto others as they give unto your simplifier."

The substance of all the above is that simplifiers vary because they are designed to run along side programs which do very different jobs from each other. The integrator is concerned with elementary algebraic expressions and uses the elementary rules learned in a first course in algebra. When the routine calling the simplifier requires some special form, then that form should be supplied outwith the simplifier. In view of the above, a definition of "algebraic simplification" is proposed: Algebraic simplification is the set of rules for consolidating an expression to its most concise and usable form while preserving the structure of the originally supplied expression.

4.2 The organisation of the simple man's simplifier

The organisation of the simplification routine in IDEA is devised to incorporate the best balance of simplicity, ease of implementation, and speed of computation. The simplification problem is approached remembering that expressions for input are in tree form. A tree is recursively defined as a network of smaller trees, so each sub-tree may be treated in the same way as the whole tree. A recursive routine becomes the best way to handle each of the sub-trees, and then the main tree.

The rules of simplification are chosen from experience, most of the tips one would expect to pick up in a first course on algebra are incorporated into IDEA. Each rule is applicable to a given algebraic operator, so the rules grouped under the operator to which they refer. It is easy to find the rules which might be applicable to a given expression

by examining the top operator of its tree representation.

Since there are only a few algebraic operators to find, a linear search method is adequately fast and easy to implement. Monadic functions, except for the negative sign, have no special simplification rules, e.g. expn(ln x) = x, implemented due to both size and time limitations. Functions are handled by ensuring only that their arguments are in simplest form. Auxilary procedures help the main routine recognise some difficult patterns and apply refinements.

4.3 The simplification rules and their implementation

4.3a The auxilary routines

Several routines aid the main simplification program in its purpose: to compact the inputted expression to an easily readable form while preserving the form of the inputted expression.

It is a problem in simplification to recognise expressions as being equivalent when the commutative properties of addition and multiplication have been applied. For example, when treating expressions as a rather anonymous list of symbols, it is an arduous task to show a machine that, in context, a+b*c is the same as c*b+a. One solution to the problem is to canonically order expressions on input so that equivalent expressions are always represented identically internally. However, then it becomes difficult to return to the original form of the expression, which is shown above to be desirable. Changes in the form of the original expression may have side effects as is demonstrated in the $(x+a)^n$ example above. Thus, the simplifier itself is provided with a method of deciding whether or not two expressions are equivalent given only the expressions themselves. The method is incorporated in the subroutine "equal." If the top operator of an expression is addition or multiplication, the operand arms of the

expression are "checked off" against those of the expression to which it is being compared. If each operand has a matching equivalent operand in its opposite number then the expressions are deemed equivalent.

Subroutine "gcd" computes the greatest common divisor of two numbers by Euclid's method.

"Cancel" is a routine to help in the simplification of divisions. It divides out common factors of numerator and denominator of a fraction taking into account the commutative property of multiplication. It does this by checking each operand arm of a multiplication in the numerator against each operand arm of multiplication in the denominator. "Subtract" is a subroutine which is similar to "cancel", but which acts upon subtraction instead of divisions. It ensures that examples such as a+x-a and a+x+y-a are simplified to an expression which contain no a's.

It is decided that all expressions which contain divisions should be put over a common denominator. The procedure "comdenom" finds that denominator by multiplication of common factors.

All these routines, and some of the other more trivial ones the reader may find in appendix A, use algorithms similar to those used by human algebraists to achieve their ends. They all preserve form as much as possible.

4.3b The simplification rules

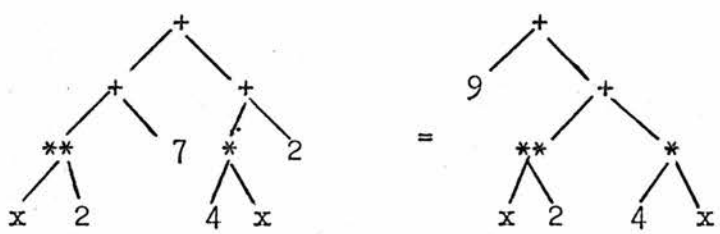
The simplification rules are grouped under the operator to which they apply. Simplification occurs recursively for each subtree in the expression ending when an atomic node is found, i.e. a number or variable standing alone. Monadic operators merely have their arguments simplified to zero and a negative of negative is made a positive. The simplification rules for exponentiation, subtraction, addition, division, and multiplication are

tabulated in table IV-1 at the end of the chapter.

4.3c A refinement routine

There is a convention in the IDEA simplifier that brings all numbers as far up and to the left of the tree as possible. This ensures that numbers are in an easy place to find and to manipulate. This is done using the routine "collect" and the associative law over addition and multiplication. This situation is illustrated by example.

Example of number grouping



Rules for exponentiations

$x^0 = 1$
$1^x = 1$
$x^1 = x$
numbers raised to integerial powers are multiplied out
$x^{a^b} = x^{ab}$
$(-x^{2n}) = x^{2n}$
$\left(\frac{x}{y}\right)^n = \frac{x^n}{y^n}$
$(expn(a))^b = expn(ab)$

Rules for subtractions

if the operands are numbers, perform the subtraction
x-0 = x
0-x = -x
x-x = 0
if expression contains fraction, find common denominator
-x-(-y) = y-x
x-(-y) = x+y
x-(p*x) = (1-p)*x
(a*x) - (b*x) = (a-b)*x
(a*x)-x = (a-1)*x

Table IV-1 (continued on next page)

continuation of Table IV-1

Rules for divisions

$x/1 = x$
$0/x = 0$
divide out the greatest common
denominator of fractions
$x/x_n = 1$
$x/x^n = x^{1-n}$
$x/(y/z) = (x*z)/y$

$x^n/x = x^{n-1}$
$(x/y)/z = (x*z)/y$
$x^n/x^m = x^{n-m}$

Rules for multiplications

if the operand arms are numbers,
perform the multiplication
$x*0 = 0$
$x*1 = x_2$

$x*x = x$
$-x*-y = xy$
$x*x^n = x^{n+1}$
$x^n*x^m = x^{n+m}$

Rules for additions

If the operand arms are numbers,
perform the addition
$x+0 = x$
$x+x = 2*x$
If an operand arm is a fraction, put the
expression over a common denominator
$-x+-y = -(x+y)$
$x+-x = 0$
$x = -y = x-y$

Rule found in all classes; bring negatives to the outermost level,
e.g. $x/-y = -(x/y)$, $x*-y = -XY$, etc.

CHAPTER V
INTEGRATION

5.1 The executive program

The subroutine "int" controls the integration process. First, it simplifies the integrand, which is supplied to the procedure in polish tree form, using the routine described in Chapter IV. This eliminates superfluous expressions in the integrand. It is anticipated that the user is not so likely to supply an unsimplified expression so much as an integration routine recursively calling "int" (see below).

The simplified integrand is matched against the integral tables by calling the subroutine "basicform." The entries in these tables form appendix D. The organisation and workings of "basicform" are the subject of section 5.2.

If the integrand cannot be found in the integral tables, then the integrand is passed to any of several routines, depending on the top operator of its polish tree, which returns as its value a list of integration functions. These decision routines are described in section 5.3. The integration functions are described in section 5.4.

The procedure "apply" operates the integration procedures on the integrand in the order supplied in the suggestion list. Each integration does its work on the integrand and recursively calls "int" to start the whole procedure again, the next time with the newly transformed integrand. Thus, a depth-first heuristic search is implemented.

Before describing the integration procedures outlined above, it is important to discuss the philosophy with which these integration procedures are written. The process of integration is viewed here as a process of simplification in that integrands are changed to equivalent forms with the goal in mind of creating a form in the integral tables. The transformation process continues always with a reference point, a context, namely the

variable of integration. Unlike the simplification procedure described in chapter IV, the integration procedure has two tools to transform the integrand: it may be transformed by actually changing the integrand itself algebraically to some more useful form, or the variable of integration may be changed to give a different context to the integrand. In the latter case, transformation of the integrand itself is usually necessary to ensure that the original context of the problem is not lost.

The astute reader will already have noticed that any expression which does not explicitly contain the variable of differentiation will be considered constant, i.e. it will differentiate to zero. The variable of integration or of differentiation is normally thought of as being some single name such as "x" or "t." The organisation of IDEA is such that a whole tree may be used as the "variable" of integration or of differentiation,

Thus, if the polish tree representation of the integrand or expression to be differentiated does not explicitly contain the "tree of differentiation," it will be considered relatively constant.

Examples

$$\frac{d}{dx}\ (y) = 0$$

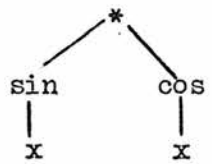
$$\frac{d}{d\ (\tan x)}\ (\tan(x^2)) = 0$$

$$\frac{d}{d(x^2)}\ (\tan x) = 0$$

$$\frac{d}{d(\tan x)}\ (\tan(x)) = 1$$

$$\frac{d}{dx}\ (\tan\ (x^2)) = 2x\ \sec^2(x)$$

Before going on to a discussion of the concrete advantages of this treatment of differentiation, it would be well to calm the fears of those mathematicians who might object to this approach. Some mathematicians might object, for example, that in the second example above, $\tan(x^2)$ is not sufficiently constant with respect to $\tan(x)$ that differentiating one with respect to the other should not become zero as it does in IDEA. This objection would become serious if it were expected that people would actually request a differentiation with respect to some formula. But it is expected that only the integrator would make such a request, and only if the integration at hand becomes any easier by assuming such a representation. In section 5.3a, it is shown how the heuristics make a decision to apply change-of-variable substitution to such an integrand as



since differentiation of the left subtree, with respect to x, produces the right subtree. But if the right subtree were any more complicated than it is here, say by containing a $\cos(x^2)$, then the substitution is not made and $\sin(x)$ does not become the new variable in integration. Thus, IDEA does not allow itself to be misled by its own differentiator. Examples of integration using formulae as the variable of integration are posed by authors of calculus texts, e.g. Ayres.[2]

This approach has several advantages. First, it helps decide which expressions are constant compared to which other expressions; merely differentiate one with respect to the other and see if the answer simplifies

to zero. Second, certain transformations, especially change-of-substitution become very easy. For example, if

$$\int \sin x \cos x \, dx$$

is considered, then let y=sin x and dy=cos x dx. Note that the dx is more than an indication of which variable is to be considered in the integration. It is part of the expression itself and serves to indicate completion of the chain-rule.

$$\int f(x)\, dx \quad .$$

really means given f(x) times the derivative of its argument á lá chain rule, find the expression which differentiates to this expression. So when a change-of-variable substitution as in the above example is made, instead of changing the expression from one in x to one in y, the sin(x) is left alone and the variable of integration is changed to sin(x).

$$\int \overbrace{\sin x}^{y} \quad \overbrace{\cos x \, dx}^{dy=d(\sin x)} = \frac{\sin^2 x}{2}$$

Using the approach here, no back substitution for y is necessary in IDEA. The answer is printed out immediately. The change-of-variable routine is outlined in section 5.4b, and there a disadvantage of this approach is discussed.

In the routines described in the following sections of this chapter, a given substructure in an integrand is operated on. The substructure might be eliminated, as in factorisation of constant, or the variable of integration might be enlarged into some larger tree, as in change-of-variable substitution,

or it might be replaced by some other more usable structure, as in integration by parts. The first two types of methods physically reduce the number of substructures in the integrand or increase the size of the variable of integration so that the integrand is smaller in context. Thus, as long as an integrand containing a finite amount of substructures is supplied, the integration procedure terminates either in an answer, or failure when the integrand loses all its substructures or when none of the remaining substructures can be operated upon by a known method. In the last type of method, the transformed integral must be double-checked by some means to ensure that some progress towards solution is being made, otherwise infinite search occurs.

The executive procedure carries out some transformations itself. Integrals which are sums are integrated according to the rule

$$\int v \pm u \, dx = \int v \, dx \pm \int u \, dx$$

Although this rule is appropriate for a great many problems, it is not always appropriate. An example would be

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

The first term is unintegrable in terms of elementary functions so the total integration fails. The only method for handling this case is creating a heuristic guess at the answer such as Moses did in implementing the Risch algorithm.[26] These methods are time consuming and large and hence not included in IDEA. A sharp eye sees that the answer to the above integration is just the reverse of the differentiation rule for multiplication, viz. xe^{x^2}.

The simplification routine puts negative signs at the top of the tree.

Thus "int" performs the transformation

$$\int -u \, dx = -\int u \, dx$$

with ease.

5.2 The integral tables

The entries in IDEA's integral tables are similar to those in Ayres.[2] The types of functions which are entered in the tables are catagorised as exponentiations, multiplications, divisions, square roots of functions, or as monadic functions. The first four catagories are examined by separate IDEA procedures, while the monadic functions are examined by the integral table look-up executive procedure "basicform."

5.2a "expform"

The subroutine "expform" tests exponentiations (not to the $\frac{1}{2}$ power which is a square root form and examined elsewhere, cf. 5.2d) against the following exponential entries:

$$\int \sec^2 x \, dx = \tan x$$
$$\int \csc^2 x \, dx = -\cot x$$
$$\int a^x \, dx = \frac{a^x}{\ln a} \quad a \text{ is a constant}$$
$$\int x^n \, dx = \frac{x^{n+1}}{n+1} \quad n \neq 1$$

Remember that $e^{f(x)}$ is handled as the function expn(f(x)) and not as an exponentiation. This point is discussed further in Chapter VI.

5.2b "multform"

The IDEA procedure "multform" handles integral table entries of the following forms:

$$\int \text{Sec } x \tan x \, dx = \text{sec } x$$
$$\int \text{csc } x \cot x \, dx = -\text{csc } x$$

5.2c "divform"

The procedure "divform" is a bit more complicated than those described in sections 5.2a and 5.2b. It handles the table entries which are divisions. The order of description here of the table entries follows, as closely as possible, the order of entries actually used in the subroutine.

Remembering that functions which differentiate to zero are considered constant, the first possibility tried is

$$\int \frac{a}{b}\, dx = \frac{ax}{b}, \quad a \text{ and } b \text{ constant}$$

All further entries in this section of the table are unity divided by something else. If the numerator is not unity, a quick exit is afforded from the routine.

With the numerator taken care of, the denominator is concentrated upon. IDEA first examines denominators which are additions or subtractions. All such entries are of a form containing a polynomial in the variable of integration in the denominator. IDEA's subroutine "polynomial" ascertains this fact and returns the degree of the polynomial and a list of the coefficients of the terms of the polynomial. This is helpful because the only type of denominator considered here is a second degree polynomial whose first degree term is zero. They are listed here:

$$\int \frac{dx}{a^2x^2+b^2} = \frac{1}{ab} \arctan \frac{ax}{b}$$

$$\int \frac{dx}{a^2x^2-b^2} = \frac{1}{2ab} \ln \frac{ax-b}{ax+b}$$

$$\int \frac{dx}{b^2-a^2x^2} = \frac{1}{2ab} \ln \frac{ax+b}{ax-b}$$

The next sort of denominator considered by "divform" is a square-root type of denominator.

$$\int \frac{dx}{\sqrt{ax + b}} = 2\sqrt{ax + b}$$

$$\int \frac{dx}{\sqrt{a^2x^2 \pm b^2}} = \frac{1}{a} \ln\left(ax + \sqrt{a^2x^2 \pm b^2}\right)$$

$$\int \frac{dx}{\sqrt{b^2 - a^2x^2}} = \frac{1}{a} \arcsin\left(\frac{ax}{b}\right)$$

After these the only case left to recognise is

$$\int \frac{dx}{x\sqrt{a^2x^2 - b^2}} = \frac{1}{a} \arcsin\left(\frac{ax}{b}\right)$$

5.2d "sqform"

The next part of the integral table is the table of square-root forms, i.e. those expressions raised to the $\frac{1}{2}$ power. The forms considered are those which are roots of second degree polynomial whose first degree term is zero.

$$\int \sqrt{a^2x^2 + b^2}\, dx = \frac{1}{a}\left[\frac{a}{2} x\sqrt{a^2x^2 + b^2} + \frac{b^2}{2} \ln\left(ax + \sqrt{a^2x^2 + b^2}\right)\right]$$

$$\int \sqrt{a^2x^2 - b^2}\, dx = \frac{1}{a}\left[\frac{a}{2} x\sqrt{a^2x^2 - b^2} - \frac{b^2}{2} \ln\left(ax + \sqrt{a^2x^2 - b^2}\right)\right]$$

$$\int \sqrt{b^2 - a^2x^2}\, dx = \frac{1}{a}\left[\frac{a}{2} x\sqrt{b^2 - a^2x^2} + \frac{b^2}{2} \arcsin\left(\frac{ax}{b}\right)\right]$$

5.2e The defined functions

The unary functions defined for integration in IDEA are tabulated in the executive program "basicform." These include the trigonometric functions, the logarithmic functions, and the expn function.

The strategy used in "basicform" is the same as the strategy used by much of the rest of the integration routines. Classification of an integrand begins with the examination of the top operand on its polish tree. Further classification takes place by examining substructures below the top operand. By searching for more general characteristics before more specific ones, the search proceeds rapidly and the program runs quickly. In "basicform," it takes but four or five levels of search before a positive decision is made about whether or not the integrand corresponds to an integrand in the tables.

5.3 The heuristic functions

If a given integrand cannot be found in the integral tables, then intelligence, or in this case artificial intelligence, must be used to seek a solution. This section deals with the ways in which IDEA deduces just what to do. The basic routine is that the integrand is examined to see if it contains substructures which, in IDEA's vast "experience" in integration, indicate that a given method or transformation might work.

It might be argued that if the integral tables were only large enough, then the sort of solution described in this section and the following would become unnecessary. There are serious questions as to just what is meant by "large enough." If entries are added to the integral table when each new case is imagined, the integral table becomes very overweight and has a large search time. By generalising cases into groups for which given methods have a high probability of being applicable, many cases are solved for which several entries in the integral tables would have been needed. For instance, integrands which contain $\sqrt{a^2x^2 - b^2}$, $\sqrt{a^2x^2 + b^2}$, or $\sqrt{b^2 - a^2x^2}$ can often be done by trigonometric substitution. Forms containing $\sqrt{ax^2 + bx + c}$ can often be done this same way once the square is

completed. But exactly which of the first three forms above this last form will look like is not obvious upon inspection. Furthermore, this last form often requires a change-of-variable substitution after the square has been completed. Covering all these cases would be difficult in a table, yet they are all handled by IDEA's one procedure "trigsub." A careful balance must be made between the size of integral table and the amount of heuristic methods of integration.

IDEA uses a depth-first method of heuristic search. Method generation, described below is done carefully so that methods which are most likely to lead to a solution are generated first and therefore tried first by "apply." The premise is that methods which eliminate part of the expression tree, explicitly or implicitly, are more likely to help than those methods which feel for an answer heuristically. Examples of methods in the first category are factorisation of constants and change-of-variable substitution. An example in the second category is integration by parts.

Each integrand is passed to examination subroutines according to the top operator on its polish tree. Each subroutine returns a list of functions, or the null list, which are deemed applicable to the integrand. If factorisation of constant or change-of-variable substitution is deemed applicable, then the routine exits as soon as practicable since these methods are so useful in general that they should be started with minimum delay. Otherwise, a full compliment of suggestions is built up in the variable "x."

5.3a "intmult"

"Intmult" is the IDEA procedure which produces a list of possible methods of integration for integrands which are multiplications. Since the simplifier puts numbers in the top leftmost position of the expression tree, it is a fairly easy matter to see if one is there and return the function

"facconst" along with the constant in such a case. Since people normally write down numbers and other constants to the left and the parser in turn puts these things on the left (people write "ax" instead of "xa"), then non-numerical constants are also found. If a person nests a constant for some reason, the simplifier will not bring it up front unless that constant is numerical. These sort of expressions are therefore much more difficult to recognise. It would take a bit of work to seek out a constant which was nested arbitrarily deep. The program is kept small and basic, and this sort of work is not done by IDEA.

Change-of-variable substitution is suggested for the forms

$$\int f(x)\ \ f'(x)\ dx$$

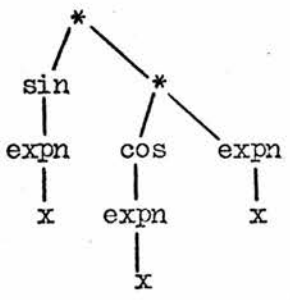
This useful method solves many integrations.

Example of integrations solved by change-of-variable substitution

$$\int \sin x\ \ \cos x\ dx$$

$$\int (\sin(\mathrm{expn}(x))\ \ \cos(\mathrm{expn}(x))\ \ \mathrm{expn}(x)\ \ dx$$

Note that the tree structure for the last expression looks like



Differentiating the topmost left subtree of the expression produces a tree which looks like the rightmost top subtree. This is the key to deciding whether or not to use change-of-variable substitution. If differeniating

one branch of a multiplication tree duplicates the other branch, save perhaps for a constant, then the change-of-variable is warranted.

"Intmult" also looks for integrands of the form

$$\int f\ (g(x))\ \ g'(x)\ \ dx$$

This form is different from the above form in that a substitution is sought for a function within a function whereas in the above case a function is multiplied by its differential.

"Intmult" returns change-of-variable for integrands containing $a^{f(x)}$, where a is a constant.

Integrands containing trigonometric functions (these are detected by the procedure "trigonometric") are candidates for the "intrig" trigonometric integration routine described in section 5.4k. Those containing other functions are suggested for integration by parts. Integrands containing polynomials raised to some power are candidates for multexpand.

"Specialform2" enables "intmult" to detect integrands which contain radical roots nested as arguments of some monadic function. "Specialform3" detects radicals not under a monadic function. These functions are described along with the routines that operate on these special integrands in sections 5.4i and 5.4j.

5.3b "intdiv"

"Intdiv," the IDEA subroutine which returns as its value a list of possible integration functions for integrands which are divisions, is similar in organisation to "intmult." It first looks for easy-to-find constants to factor in the numerator and denominator. It then looks to see if the denominator is a multiple of the derivative of the numerator, and vice-versa, in which case change-of-variable substitution is immediately returned.

Like "intmult," it also looks for derivatives of expressions contained in functions or exponentiations in the integrand's structures. If the denominator is a second degree polynomial in the variable of integration, then completing the square is added to the suggestion list. If both numerator and denominator are polynomials, and the degree of the numerator is greater than or equal to that of the denominator, then the polynomial long division routine "polydiv" is added to the list. Care is taken to discover combinations of the above such as in the case of

$$\int \frac{\cos x \ dx}{\sin^2 x + 2 \ \sin x + 1}$$

Here change-of-variable substitution is required since the denominator is a polynomial in the derivative of the numerator, except for a constant. The "separate" routine is suggested for integrands whose numerators are additions. This routine is discussed, as are all the routines of integration, in section 5.4. The remainder of "intdiv" looks much like "intmult."

5.3c "intfunc"

"Intfunc" is a short routine that returns a list of integration routines applicable to integrands which are monadic functions of some argument. Integration-by-parts is very useful for many such integrands and so is included at the end of the suggestion list. Otherwise, no new routines are returned.

5.3d "intexp"

"Intexp" returns a list consisting of integration methods suited to various integrands which are exponentiations. This routine handles cases of

$$\int \sec^2 (f(x)) \ dx$$

$$\int \csc^2 (f(x)) \ dx$$

as these two important forms are exponentiations. Polynomials in the variable of integration raised to an integer power are candidates for the routine "expansion."

Once the list of methods is compiled, it is passed to subroutine "apply" so that they may be tried. Notice that some methods get suggested for what seem to be very different integrands. For instance, "compsquare" is suggested for

$$\int \frac{dz}{z^2 - 2z + 3}$$

as well as

$$\int (z^2 - 2z + 3)^2 \, dz$$

Scrutiny shows that these forms are really very similar; the first looks much like the second of it is written

$$\int (z^2 - 2z + 3)^{-1} \, dz$$

Why the user should use one form over the other is left to him. It is up to IDEA to discover the relation.

5.4 Description of the methods of integration

The methods of integration used by IDEA and described here are by no means all the methods of integration there are. Indeed, it would probably be impossible to propose such a list and worse to implement the list on any known computer by its sheer size. It is proposed that the methods implemented in IDEA are adequate in number and variety to solve a wide variety of integrations. Each method consists ofaprocedure, or of a set of procedures controlled by some executive routine.

Certainly, all these functions of integration have less in common with each other than did the heuristic functions which creates the lists of these functions. There are, however, some features in common. They all operate on the integrand with respect to the variable of integration. In many cases, other parameters helpful to the routine are supplied by the heuristic function which suggested it. All the procedures, once they have finished their transformations of the integrand, recursively call the executive procedure "int." These are the common characteristics of the integration routines.

5.4a "facconst"

"Facconst" takes integrals of the form

$$a\int f(x)\ dx$$

and transforms them to

$$a\int f(x)\ dx$$

This is easy because the integrand classification function which suggests this method at the same time supplies the constant to be factored as a parameter.

5.4b "diffdiv"

This function takes integrals of the form

$$\int f(g(x))\ g'(x)\ dx$$

and transforms them to

$$f(u)\ du \Big|\ u = g(x)$$

The $g(x)$ in the formulae above is supplied to the routine as a parameter. The substitution is carried out in the manner described in section 5.1, viz. the integrand is divided by $g'(x)$ and $g(x)$ actually becomes the new variable of integration.

Care must be taken in change- of-variable integration in some integrands containing trigonometric expressions and some other types of integrands as

well. If the problem set to IDEA is

$$\int \frac{\sec^2 x \; dx}{\sec^2 x - 3 \tan x + 1}$$

then the change of variable $z = \tan x$, $dz = \sec^2 x \, dx$ is warranted. Making these changes produces the new integral

$$\int \frac{dz}{z^2 - 3z + 2} \Bigg| \; z = \tan x$$

which can be integrated using the complete-the-square method (cf. sec. 5.4e) but only after the trigonometric identity.

$$\sec^2 x = \tan^2 x + 1$$

has been applied. Otherwise the $\sec^2 x$ in the denominator may be considered constant with respect to the new variable of integration, tan x, when in fact it is not. IDEA takes care of this anomaly be calling the trigonometric identity routine "convert."

5.4c "intparts"

"Intparts" is IDEA's integration-by-parts routine. The two most pressing obstacles in writing a workable routine are (1) partitioning the integrand and (2) once a partition has been made, deciding whether the new integral is any easier than was the original. The former problem is handled by the associated procedure "partitions." "Partitions" is a heuristic routine which returns a list of zero or more possible partitions. These are tried out in the order generated by the routine "try." "Try" calls a procedure "sameform" which decides whether or not a given partition is useful. These routines are now described in detail.

The procedure "partitions" heuristically forms a list of ordered pairs of partitions. Each ordered pair is, respectively, the u and dv in the familiar integration-by-parts formula

$$\int u\,dv = uv - \int v\,du$$

If the original integrand is a monadic function, then u = integrand, and dv = 1 is returned as the only partition advised as, since the integrand could not be integrated as it stood, then the only other thing to try is differentiation to see if that yields a better form. If the integrand is a multiplication, several possibilities arise. For the purposes of this routine only, the parse tree is canonically ordered a bit so that monadic functions appear as far to the left as possible. (Remember that constant numbers which might otherwise be in this position have been factored out by this time.) If both the left subtree and the right subtree are more complicated than a simple atom (an atom is a variable name or a number), then the two branches become the divisions themselves. This might seem simplistic, but experience shows that this method is, for some reason, effective if an integration containing so many substructures can be integrated at all.

Examples

$$\int (ab)\ (cd)\ dx$$

yields divisions

$u = ab$ and $u = cd$
$dv = cd\,dx$ $\quad$ $dv = ab\,dx$

$$\int x(x+1)^{1000}\,dx$$

yields divisions

$u = x$ $\quad$ $u = (x+1)^{1000}$
$dv = (x+1)^{1000}dx$ and $dv = x\,dx$

The function ln and the inverse trigonometric functions differentiate to purely algebraic functions, and for this reason choosing bits of the integrand to be "u" is generally good policy.

Example

$$\int x \ln x \ dx$$

$$u = \ln x \qquad dv = x\,dx$$
$$du = \frac{dx}{x} \qquad v = \frac{x^2}{2}$$

$$\int x \ln x \ dx = \frac{x^2 \ln x}{2} - \int \frac{x}{2} \ dx$$

If the left branch is a trigonometric function, then in general it will be easy to integrate by itself and this becomes a good choice for dv.

Example

$$\int x^3 \ \sin(x^2) \ dx$$

use $u = \frac{x^2}{2}$ and $dv = 2x \sin(x^2)\,dx$

This example serves to clarify the division used in creating the partition. Without the division, the dv part would be difficult to integrate to v. The same sort of partitioning occurs if the left branch is the expn function. If both branches are exponentiations, the above rules of thumb still apply, but the various cases are nested a level deeper and hence are a bit more difficult to search. If only one of the branches is an exponentiation, then differentiating it will hopefully give an expression which is simpler since the exponent would be reduced. Integrands which are divisions can be quite difficult to partition, and there are only a few rules worth implementing. If the denominator is an exponentiation, then the derivative of it is divided into the numerator and chosen as "u" while whatever is leftover becomes "dv." This reduces the exponent in the denominator. The numerator is chosen as "u" only if both it and the denominator are exponentiations. This is simplistic, but handles a wide variety of cases while avoiding many blind alleys. Finally, if the integrand is an exponentiation, then choosing the whole integrand as "u" seems a good choice as the differentiation to du

will yield an expression with a lower exponent. Another good partition which works especially well for cases with integer exponents is dividing the function as in the following illustration:

$$\int (f(x))^n \, dx$$

$$u = f(x) \quad dv = (f(x))^{n-1} \, dx$$

This often times creates a situation where change-of-variable substitution is appropriate as in the example

$$\int (x^2 + 1)^{1000} \, dx$$

Otherwise the routine may create a recurrence relation which recursively goes through the same process until the exponent is zero at which time a different method will be tried which will lead to the answer or to failure.

"Sameform" checks to see if the integral

$$\int v \, du$$

is any easier than

$$\int u \, dv$$

The rules-of-thumb used to decide this are simplistic, but effective in many cases. Parts chosen as u must not differentiate to a monadic function if u is already a monadic function. If dv is a monadic function, then if v is not, the partition is deemed appropriate. If v is monadic, then the partition is deemed appropriate only if u is a simple polynomial in the variable of integration (which, it is hoped, could be simplified bydifferentiation to du and combined neatly with v). If both u and v are exponentiations, the worst case is that recursion occurs until one of the exponents is zero, and this is allowed.

The reader might have noticed that the "sameform" routine does not allow procedure for cases like these famous examples:

$$\int e^{ax} \sin(bx)\, dx$$

$$\int e^{ax} \cos(bx)\, dx$$

In the first example, let $u = e^{ax}$ and $dv = \sin bx$. Then $du = ae^{ax}\, dx$ and $v = -\frac{1}{b} \cos(bx)$ so

$$\int e^{ax} \sin(bx)\, dx = \frac{-e^{ax} \cos(bx)}{b} + \frac{a}{b} \int e^{ax} \cos(bx)\, dx$$

The integral on the right does not look much better than the one on the left, so one might be inclined to give up. However, if integration-by-parts is done again

$$u = e^{ax} \qquad dv = \cos(bx)\, dx$$

$$du = ae^{ax}\, dx \qquad v = \frac{1}{b} \sin(bx)$$

$$\frac{a}{b} \int e^{ax} \cos(bx)\, dx = \frac{a}{b} \left[\frac{e^{ax} \sin(bx)}{b} - \frac{a}{b} \int e^{ax} \sin(bx)\, dx \right]$$

$$= \frac{a(e^{ax} \sin(bx))}{b^2} - \frac{a^2}{b^2} \int e^{ax} \sin(bx)\, dx$$

Combining this last equation with the one above and equating sides (assuming evaluation between limits)

$$\int e^{ax} \sin(bx)\, dx = \frac{ae^{ax} \sin(bx) - be^{ax} \cos(bx)}{b^2 + a^2}$$

A similar formula is yielded in a similar way for the other problem suggested. These two famous cases are detected by the executive "intparts" routine and are just answered as if they were in the tables. This method of solution is suggested as an improvement to IDEA in chapter VI.

5.4d "arcsub"

"Arcsub" is IDEA's way of dealing with functions which are rational in sin or cos. The substitution $u = 2 \arctan z$ will replace any rational function of sin u and cos u by a rational function of z since

$$\sin u = \frac{2z}{1 + z^2} \qquad \cos u = \frac{1 - z^2}{1 + z^2} \qquad du = \frac{2\,dz}{1 + z^2}$$

The substitution $z = \tan \frac{1}{2}u$ returns to the original variable of integration. Some texts call this "substitution of tangents of half-angles."

5.5e "compsquare"

"Compsquare is IDEA's complete-the-square method in integrations of the form

$$\int \frac{dx}{ax^2 + bx + c}$$

The ax^2 x bx + c is converted to an equivalent form which looks like $(x - m)^2 + n$ where m and n depend on a, b, and c. This new form is substituted for the original and the routine also carries out the change-of-variable substitution.

$$y = x - m \qquad dy = dx$$

Generally, this new integrand will be found in the section of the integral tables containing

$$\int \frac{dx}{x^2 + a^2} \qquad \int \frac{dx}{x^2 - a^2} \qquad \int \frac{dx}{a^2 - x^2}$$

5.4f "expansion"

If a polynomial is raised to an integrial power, then expanding that expression out to all its terms and then integrating termwise may be the best method of integration. "Epansion" does just this with the aid of "expand," which applies to binomial expansion theorem:

$$(a + b)^n = \binom{n}{0} a^n + \binom{n}{1} a^{n-1} b + \ldots + \binom{n}{n} b^n$$

where

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

If the polynomial has more terms than just two, then they all will be nested in the left and right branches of the main tree, and thus if the integration cannot be done after the first expansion, the routine may later on call "expansion" again on the exponentiations that are left. Obviously, this does not happen unless absolutely necessary.

5.4g "separate"

If a sum is the numerator of an integrand which is a division, then separating the terms as

$$\frac{a_1 + a_2 + \cdots + a_n}{b} = \frac{a_1}{b} + \frac{a_2}{b} + \cdots + \frac{a_n}{b}$$

may produce a number of simple integrations which can be done separately and later added together. "Separate" accomplishes this.

5.4h "multexpand"

If the integration is of the form

$$a(b_1 + b_2 + \cdots + b_n)^m \, dx$$

then expanding the polynomial over m and multiplication of each resulting term by "a" may result in several easy-to-integrate subexpressions which may be added together to form the full anti-derivative. "Multexpand" does this.

5.4i "nest"

"Nest" is IDEA's way of dealing with integrands which contain

$$f(x^n)$$

where n may be whole or fractional, but not one, which is trivial. Such forms are detected for the heuristic routines by "specialform2," which returns the expression for which the substitution is to be made. A private variable "x¬" is used by this routine for ease of implementation. It avoids clobbering the user's variable of integration if it happens to be the same as the variable used for the substitution. The item for which the substitution is made should be the variable of integration itself.

If the exponent n is a whole number, then the substitution "x¬" = $x^{1/n}$ is made to give f("x¬"). The rest of the expression is divided by the derivative of "x¬" which is obtained by explicit differentiation.

Example

$$\int x^3 \sin(x^2)\ dx = \int \tfrac{1}{2} y \sin y\ dy \ \Big|\ y = x^2$$

For fractional n, i.e. for roots, the procedure is similar.

Example

$$\int \cos(x^{\frac{1}{2}})\ dx = \int 2y \cos y\ dy \ \Big|\ y^2 = x$$

The variable y is used in place of the private variable used in the routine in the above examples to improve readability. The fractional root case changes variables of integration by implicitly calculating dy, and then substituting $y^{(1/n)} = x$ so that the transformation remains smooth.

5.4j "radsub"

"Radsub" is IDEA's procedure for dealing with integrals which contain the form $(ax + b)^{1/n}$. Such integrands are detected by the subroutine "specialform3" which also returns the n, a, and b in the expression above. As with the nest, a private variable "x " is utilised for ease of implementation. Once a substitution for the radical has been made, viz. (au + b) = "x¬"^{n}, other occurences of the variable of integration in the integrand are

substituted by

$$\text{var} = \frac{\text{"x¬"}^{1/n} - b}{a}$$

Example

$$\int \frac{dx}{x(1-x)^{\frac{1}{2}}} \qquad \text{Let } 1-x = \text{"x¬"}^2$$

$$x = 1 - \text{"x¬"}^2$$

$$dx = -2(\text{"x¬"})\ d\text{"x¬"}$$

giving $$\int \frac{-2\ d\text{"x¬"}}{1 - \text{"x¬"}^2}$$

5.4k "trigsub"

Integrands which contain $(a^2 - b^2u^2)^{\frac{1}{2}}$, $(a^2 + b^2u^2)^{\frac{1}{2}}$, or $(b^2u^2 - a^2)^{\frac{1}{2}}$, or integer powers of these may be transformed into a trigonometric integral by one of the following transformations:

For	use	to obtain
$(a^2 - b^2u^2)^{\frac{1}{2}}$	$u = \frac{a}{b} \sin z$	$a(1 - \sin^2 z)^{\frac{1}{2}} = a \cos z$
$(a^2 + b^2u^2)^{\frac{1}{2}}$	$u = \frac{a}{b} \tan z$	$a(1 + \tan^2 z)^{\frac{1}{2}} = a \sec z$
$(b^2u^2 - a^2)^{\frac{1}{2}}$	$u = \frac{a}{b} \sec z$	$a(\sec^2 z - 1)^{\frac{1}{2}} = a \tan z$

The integration then yields an answer in z. The routine "resub" substitutes the original variable back by means of drawing a right triangle from the substitution and then using ratios of the sides to make each of the six trigonometric functions in the usual way. The routine "constants" aids simplification by grouping the constant factors produced in the integration outside the integral. Integrands which contain expressions like $(au^2 + bu + c)^{\frac{1}{2}}$ may also be integrated in thisway provided that the square is completed first. For this purpose, "comsq" is available to this

integrable form, although if the function involved is sec or csc then no further attempt is made on the integration since all the possibilities for these are tried in other routines. If m is less than zero, then a trigonometric identity is applied to make m larger than zero. This is easy since the reciprocals of trigonometric functions are also trigonometric functions. Finally, if m is larger than 2, one of the following reduction formulae is used:

$$\int \sin^m x\,dx = \frac{-\sin^{m-1} x \cos x}{m} + \frac{m-1}{m} \int \sin^{m-2} x\,dx$$

$$\int \cos^m x\,dx = \frac{\cos^{m-1} x \sin x}{m} + \frac{m-1}{m} \int \cos^{m-2} x\,dx$$

$$\int \tan^m x\,dx = \frac{\tan^{m-1} x}{m-1} - \int \tan^{m-2} x\,dx$$

$$\int \cot^m x\,dx = \frac{-\cot^{m-1} x}{m-1} - \int \cot^{m-2} x\,dx$$

$$\int \sec^m x\,dx = \frac{\sec^{m-2} x \tan x}{m-1} + \frac{m+2}{m+1} \int \sec^{m-2} x\,dx$$

$$\int \csc^m x\,dx = \frac{-\csc^{m-2} x \cot x}{m-1} + \frac{m-2}{m-1} \int \csc^{m-2} x\,dx$$

It is noticed that all these reduction formulae involve a recursive integration on $\text{trig}^{m-2} x$ so the exponent may be reduced by two and the expression given back to the integrator without even knowing exactly what "trig" is! Recursion ends when the exponent gets down to either two or one.

"Trigdiv" handles trigonometric integrands which are divisions.

If the numerator is an atom and the denominator consists of the trigonometric function on its own, a reciprocal identity is applied to trig to bring it into the numerator.

Example

$$\int \frac{x\ dx}{\sec^2 x} = \int x \cos^2 x\ dx$$

Failing this, "pattern" (described below) is called to see if the integrand is some familiar pattern which can be integrated. As a last resort, the trigonometric identity function "convert" is called to put all the trigonometric expressions in the integrand to sin or cos in which case a method like "arcsub" (see section 5.5d) might help.

"Trigmult" is the routine which handles trigonometric integrands which are multiplications. It looks for a familiar pattern by calling "pattern" (described below). If not all the components of the integrand are trigonometric, integration by parts is tried. As with "intdiv," the routine converts all the trigonometric functions to sin or cos using "convert" as a last effort to integrate.

The workhorse of both "trigdiv" and "trigmult" is "pattern." The trigonometric integrals which come up the most can be classified into a few catagories. Divisions can look like multiplications by changing them to the numerator times the denominator to a negative power.

Example

$$\int \frac{\sin^m x\ dx}{\cos^n x} = \int \sin^m x \cos^{-n} x\ dx$$

"Pattern" returns as its value either a procedure designed to handle the case it has found or zero in the case of failure. The reductions used

-strive for a form which change-of-variable integration may apply, or alternatively strives for a form which may be expanded by the binomial expansion theorem giving a sum of powers of unique trigonometric functions each of which could be handled by "intexp." If the change-of-variable option is selected, then the change of variable is effected within the particular routine. These comments are summarised in Table V-1.

Table V-1

Common trigonometric integrals with their solutions

Pattern	Reduction Used	IDEA Subroutine
$\int \sin ax \sin bx\, dx$	$= \int \frac{1}{2} \cos(a-b) \cos(a+b)\, dx$	sinsin
$\int \sin ax \cos bx\, dx$	$= \int \frac{1}{2} \sin(a-b) \sin(a+b)\, dx$	sincos^2
$\int \sin^m x \cos^n x\, dx$	$= \int \sin x\, (1-\cos^2 x)^{\frac{m-1}{2}} \cos^n x\, dx$, m odd	redsin
	$\int \cos x\, (1-\sin^2 x)^{(n-1)/2} \sin^m x\, dx$, n odd	
	else $\int \sin^m x\, (1-\sin^2 x)^{n/2}\, dx$	
$\int \tan^m x \sec^n x\, dx$	$= \int \tan^m x (\tan^2 x+1)^{(n-2)/2} \sec^2 x\, dx$, n even	redtan
	$\int \sec x \tan x\, (\sec^2 x-1)^{\frac{m-1}{2}} \sec^n x\, dx$ m odd and n odd	
	else $\int (\sec^2 x-1)^{m/2} \sec^n x\, dx$	
$\int \cot^m x \csc^n x\, dx$	$= \int \cot^m x (\cot^2 x+1)^{(n-2)/2} \csc^2 x\, dx$, n even	redcot
	$\int \csc x \cot x\, (\csc^2 x-1)^{\frac{m-1}{2}} \sec^n x\, dx$ m odd and n odd	
	else $\int (\csc^2 x-1)^{m/2} \csc^n x\, dx$	

The reader may also notice that the reduction routines mentioned in table V-1 try to integrate some integrals which have fractional or negative exponents on one of the trigonometric functions involved.

Examples

$$\int (\tan x)^{\frac{1}{2}} \sec^4 x \, dx = \int (\tan x)^{\frac{1}{2}} (1 + \tan^2 x) \sec^2 x \, dx$$

$$\int (\sin x)^{-2} \cos^3 x \, dx = \int (\sin x)^{-2} (1 - \sin^2 x) \cos x \, dx$$

5.4m "polydiv"

"Polydiv" attempts to integrate integrations of the form

$$\int \frac{r(x)\, dx}{s(x)}$$

where r and s are polynomials in x and the degree of r is greater than or equal to the degree of s. The degree of the numerator r along with the coefficients of its terms are passed to the division routine along with the same information on the denominator s. The division routine follows the long division routine described by Knuth. (vol. 2).[14] A quotient of leading coefficients of numerator and denominator is produced, multiplied through the denominator and subtracted from the numerator to produce a remainder. This remainder is recursively supplied to the routine as the new numerator while the quotient of leading coefficients becomes the leading coefficient of the division. This procedure eventually produces

$$\int \frac{r(x)\, dx}{s(x)} = \int \left[p(x) + \frac{r*(x)}{s*(x)} \right] dx$$

where p is a polynomial and integrated termwise by the subprocedure "termwise," and r* and s* are polynomials with the degree of r* is less than the degree of s* and can hopefully be integrated by the other methods at the disposal of IDEA.

---- ---- ----

This ends the description of the integration routines of IDEA. It is felt that the methods used here are easy to implement, easy to understand, work well together, and run efficiently. Improvements could certainly be made; these are partly the substance of the next chapter.

CHAPTER VI
COMMENTS, CONCLUSIONS, AND SUGGESTIONS FOR FURTHER WORK

6.1 Performance of the integrator

On the whole, the performance of the integrator is pleasing. Firstly, the whole program requires only 1600 cards of SASL source code. This is a considerable saving over the 70 pages of LISP code used in Moses' program at only a slight reduction in power. Furthermore, it is viewed here that SASL is a much more readable language than LISP.

The performance of the integrator is also good as far as ease of use is concerned. The amount of effort required from a user to perform a differentiation or integration is minimal. It is disappointing that the integrator could not be implemented on the timesharing facilities available to the author due to memory limitations; this would further convenience the use of the package. The differentiation package alone has been put onto the timesharing facilities of St. Andrews University to the great satisfaction of many users.

The speed at which answers are derived is also satisfactory. The complete integration package takes almost three minutes just to be put into "S-code" (intermediate code for interpretation by the SASL virtual machine). It would be a saving if SASL were constructed such that "S-code" could be stored without having to reconstruct it for each run. The same could be said for the timesharing differentation package which now takes about 30 seconds to compile. Further speed-ups would be realised if SASL were compiled direct to machine code. If more memory were available, time would be saved because fewer "garbage collections" would be necessary. As it stands, most differentiations can be completed inside of one minute (not including compilation) and most integrations can be completed inside of five minutes (also not including compilation).

The most disappointing aspect of this research from the author's point of view is the coding. The project began at a time when the author had little programming experience, and some routines are thus coded perhaps clumsily. The author has unfortunately had little time to rethink these routines. When using a language which encourages abstraction, programs are generally better when abstracted to the fullest.

Another disappointment arises from the author's frustration in solving certain systems problems (notably the implementation of SASL into a large memory) which deprived him the chance to test the program fully. While the program works well at this writing, future users may uncover as yet unforseen difficulties. This is unfortunate, but time does not allow further research.

The output of the program would be more meaningful if intermediate results berween initial problems and final answer were printed rather than just the final answer. This would give the output an appearance like that of a human solver as well as making the program valuable as a tutor. This was not done as SASL has no imperative "write" statement and working around this would be tedious as well as detracting from the problem at hand. This "write" statement could be added to the SASL language without great effort and it is suggested that research into this be carried out.

6.2 The organisation of IDEA

IDEA exhibits some programming techniques which bear emphasis because they realise a more concise, more readable program than the techniques used by a vast number of computer scientists today. Most tasks in IDEA are performed by a function, one or more functions to each task (as opposed to one or more tasks to each function). This approach is forced by the very use of SASL as a programming language because SASL programming is very

function intensive, i.e. SASL is created in such a way that it becomes necessary for the programmer to break his tasks down into functions which are evaluated by the main program call. This is not to say that this approach is not available in other programming languages; it is available in many other languages. People who do use this approach tend to be more often labelled "good programmers" whereas people who do not, while not being necessarily "bad programmers," nevertheless find many times that their programs lack clarity. The simplification routine in IDEA could be improved by breaking it down into more functions, say one function for simplifying additions, one for multiplications, and one function for every other kind of top operator on the polish tree representation. This would bring the organisation of the subroutine simplify more closely into line with that of "int" or "intrig" which are perhaps more readable routines. The readability which programs written in SASL seem so often to enjoy is also enhanced by the lack ofimperatives and assignments in the language.

The list of integration methods (functions) which are sent to "apply" by the routine "int" are sent in order of likehood of working. This order is not deduced by implementation of some great mathematical proof, it is done solely by experience. SAINT generated all the possible successors of a given integrand form first and then evaluated them as to which successor was best to continue. SIN uses tighter constraints on generation of successors and IDEA uses a very like amount of strictness. The main difference between SAINT heuristics and IDEA and SIN heuristics is that in SAINT an integrand's successors are rated heuristically for integrability while in IDEA and SIN the integrand itself is costed. IDEA especially makes great use of rules-of-thumb for deciding which method would be best by careful scrutiny of the substructures comprising the integrand.

Generally, an integrand contains at most two or three significant substructures (i. e. clue-giving substructures) and thus would generate only two or three methods, each of which would have a good probability of working. Since each method of integration reduces a significant substructure to a simpler one (with the goal in mind of creating an integrand which might be a key to the integral tables) then successor forms will generate fewer methods than their ancestors. There comes a time then when zero methods are generated for a given successor, in which case failure occurs, or an answer is found. IDEA has never been found to be infinitely recursive in its search for a solution.

The use of anonymous functions in the "diff" and "output" routines are perhaps not direct in that the anonymous function must be evaluated before an answer can be arrived at whereas it might be more direct to return the expression within the "diffop" routine itself. In "diff" the routine looks like

```
let diff s var
be  . . .
      .
      .
      .
    diffop (hd s) var (tl s)
and diffop s var
be  . . .
    s = '* -> lambda (left, right,)
                ...
    s = 'sin" -> lambda (operand,)
                ...
```

Diffop returns a function which must operate on the tail of the expression supplied to "diff." Suppose the routine were instead coded

```
let diff s var
be    .
      .
      .
    diffop s var
and diffop (head, tail) var
be    .
      .
      .
```

```
head = % * ->let (left,right,) = tail
           in
    . . .
head = 'sin" ->let (operand,) = tail
           in  . . .
```

In this case the answer to the differentiation would be returned directly from "diffop." A great deal of efficiency does not seem to be lost by the former method over the latter, but the former's coding seems a bit more elegant than the latter. In weighing this tradeoff, the former was chosen.

6.3 Improvements to IDEA

Hindsight is 20-20. It is recognised that some important aspects of the integration problem are neglected in IDEA. These omissions are discussed in this section. Suggestions for the implementation of these other integration sub-problems are also presented.

6.3a Definite integration

Most definite integrals can be computed trivially by evaluating the indefinite integral at the limits of integration. IDEA does not do this for two reasons. The first reason is that the language, SASL, has no real numbers which are necessary for a really complete implementation. This problem could be alleviated by using another meta-language, but this seems rash in view of the neat implementation of the indefinite integration SASL enables. Another way of alleviating this problem would be to represent all numbers as rational, but this would require the construction of special handling routines for these rationals and thus further tax the already strained computer system.

The second reason for not implementing definite integration in IDEA is that sub-programs for the c omputation of such important functions for calculating the trigonometric functions, the inverse trigonometric functions,

and the logarithmic functions are absent from the IDEA system. This is complicated by the fact that the implementation of such routines would have to be done in the absence of real numbers.

Forms involving complex or infinite limits of integration pose more serious problems of evaluation than those described above. In chapter I it was surveyed how Moses implemented many of these difficult forms at the cost of much computation.

6.3b Multiple integration

Multiple integration is often times a trivial extension of single integration. Thus, it could be implemented using a loop of integration on a function and a list of variables of integration one of which would be used each time through the loop. However, multiple integration seems lackluster when there is no definite integration routine to transform the integral each time through the loop. Also, such a routine would again heavily tax the resources of the computer at the gain of little facility.

6.3c Integration by transpositions

The integrator could handle more types of integrands if there were an integration-by-transpositions method. This method would be part of the integration by parts routine and would handle forms similar to

$$\int e^{ax} \cos bx \; dx$$

The process is described in section 5.4c. Timewise, an integration-by-transpositions routine might be expensive as the left and right sides of an integration-by-parts equation would have to be constantly surveyed for items which could be transposed. Furthermore, the decisions as to whether or not progress is being made at each step of the integration-by-parts process would be difficult and expensive to make.

6.3d Expressions containing "e"

Expressions containing "e", the natural base, and powers of e pose some of the most difficult problems of integration. Reduction formulae operating on the exponent of e are frustratingly difficult to work out, as all students of the classic

$$\int e^{x^2}\,dx$$

are likely to know. It becomes difficult to recognise e if it is not treated as a function (e.g. e^x = expn(x)) as is done in IDEA. Treating it as an exponent has some advantages. The irrational e is a constant and as such can be treated like any other case of a^x where a is constant. Thus rational expressions in e could be handled as in the following example:

$$\int \frac{e^x\,dx}{2 + 3e^{2x}} = \int \frac{dy}{2 + 3y^2} \quad \Big| \quad y = e^x$$

A clever detection package becomes necessary to make this sort of substitution and a good rational package (discussed further in section 6.3g) is necessary to complete the above integration. Some of the stickier integrations involving e may be obtained by the theoretical Risch decision procedure.[26] This requires much in the way of both computer coding and computer resources and is beyond the simple integration package IDEA is intended to be.

6.3e Hyperbolic functions

Integrands containing hyperbolic functions could be handled in much the same way as trigonometric functions since the identities for hyperbolics are so similar to their trigonometric analogs. A further improvement would be a routine to convert the hyperbolics to their exponential equivalent and then using the sort of package glossed over in section 6.3d. It is not

implemented into IDEA due to size and time restrictions.

6.3f Algebraic transformations

IDEA could handle a wider variety of integrands if it went through a good course in algebra. Among the sorts of things it could handle after such a course would be forms like

$$\int \frac{x\ dx}{\sqrt{x^2 + a^2} - x}$$

Multiplying both top and bottom of this by

$$\sqrt{x^2 + a^2} + x$$

gives

$$\int \frac{x(\sqrt{x^2 + a^2} + x)\ dx}{a^2}$$

and simplification and other integrations could be applied.

A very good algebraic package would handle relationships between a first degree polynomial appearing in the numerator and a second degree polynomial in the denominator. The following examples illustrate:

Examples

$$\int \frac{2x - 3\ dx}{4x^2 - 11} = \tfrac{1}{4} \int \frac{8x - 12\ dx}{4x^2 - 11}$$

$$= \tfrac{1}{4} \int \frac{8x\ dx}{4x^2 - 11} - \frac{3}{2} \int \frac{2\ dx}{4x^2 - 11}$$

$$= \tfrac{1}{4} \ln \left| 4x^2 - 11 \right| - \frac{3\sqrt{11}}{44} \ln \left| \frac{2x - \sqrt{11}}{2x + \sqrt{11}} \right| + C$$

$$\int \frac{x + 2\ dx}{\sqrt{x^2 + 2x - 3}} = \tfrac{1}{2}\int \frac{2x + 4\ dx}{\sqrt{x^2 + 2x - 3}}$$

$$= \tfrac{1}{2}\int \frac{2x + 2\ dx}{\sqrt{x^2 + 2x - 3}} + \int \frac{dx}{\sqrt{(x+1)^2 - 4}}$$

the square completed

$$= \sqrt{x^2 + 2x - 3}\ \ 1n(x+1+\sqrt{x^2 + 2x - 3}) + C$$

Both these integrations were done mostly with the aid of algebraic manipulation. Just as it is useful for the human student of calculus to know algebra before embarking on the calculus course, so too should the symbolic integrator have a good knowledge of algebra.

The expense of writing a good system is more than trivial for all the possibilities to be taken into account. It should be organised in such a way that the algebra subroutine can scan an inputted expression and make a decision as to whether or not it should operate on the expression. Such a decision would best be made by considering the substructures present in the inputted expression. If the expression contained trigonometric or other functions, an algebraic routine would be less likely to help matters than if the expression consisted only of polynomials (or the roots and powers of polynomials) joined by addition, subtraction, multiplication, or division. Just as IDEA has a routine for handling specifically trigonometric functions, so too should it have one for specifically algebraic problems.

6.3g Rational functions

There is one type of algebraic manipulation that enables such a large increase of power in an integrator that it deserves special mention. IDEA would handle a far greater domain of problems if a good rational function

routine were implemented which handled problems of the form

$$\int \frac{r(x)\,dx}{s(x)} \qquad \text{(i-1)}$$

where r and s are both polynomials.

IDEA does at present take the first step in handling this sort of problem in that if the degree of r is greater than or equal to the degree of s, then long division is carried out to obtain

$$\frac{r(x)}{s(x)} = p(x) + \frac{r*(x)}{s*(x)} \qquad \text{(i-2)}$$

where p is a polynomial (and trivial to integrate) and r* and s* are polynomials such that the degree of r* is less than the degree of s*. If IDEA cannot do the integration

$$\int \frac{r*(x)\,dx}{s*(x)} \qquad \text{(i-3)}$$

then it reports failure for the whole integration.

One sort of rational function is fairly easy to integrate of relations are noticed between the exponents. This form is best presented by example:

$$\int \frac{x^5\,dx}{x^{12}+1} = \frac{1}{6}\int \frac{dy}{y^2+1} \qquad y = x^6 \qquad \text{(i-4)}$$

The relation is worked out between the exponents of the original problem, the exponents of the expression to be substituted for, and the exponents of the derivative of the expression to be substituted for. A substitution is made if the latter two expressions can be multiplied together to form the former expression. Greatest common divisor calculations are one aid to this division.

Possibly the most difficult integration problems there are contain polynomial fractions which do not fall into any of the catagories discussed to this point. A thorough study of the algorithms for handling such forms is made by Horowitz, who also explores the computing times involved in churning through such algorithms.[12] The algorithms take integrals of the form

$$\int \frac{A(x)\,dx}{B(x)} \qquad (i\text{-}5)$$

where A and B are polynomials and the degree of B is greater than the degree of A. The algorithms find the unique factorisation of the polynomial fraction such that

$$\frac{A(x)}{B(x)} = \frac{A_1(x)}{B_1(x)} + \frac{A_2(x)}{B_2^2(x)} + \cdots + \frac{A_n(x)}{B_n^n(x)} \qquad (i\text{-}6)$$

Each $A_i(x)$ has a degree less than that of denominator $B_i^i(x)$ and each $B_i(x)$ has a degree less than or equal to two. SIN utilises the classic method of discovering this factorisation, Hermite's method, which is implemented by Manove, et al.[15] Hermite's method finds these partial sums by multiplying both sides of equation i-6 by B(x) and equating coefficients or by doing greatest common divisor calculations on A and its first derivative. Equation i-6 may be broken down farther into

$$\int \frac{A(x)\,dx}{B(x)} = \int \left[\sum_{i=1}^{k} \sum_{j=1}^{i} \frac{A_{i,j}(x)}{B_{i,j}(x)} \right] dx \qquad (i\text{-}7)$$

The $A_{i,j}$ can be deduced by equating coefficients in a rather massive matrix scheme of simultaneous equations. Horowitz in his paper introduces a new scheme of deducing this partial fraction decomposition which utilise special mathematical properties during the matrix solution which keep down the sizes

of the coefficients during the intermediate stages and shows this to have an effect on the final computation time. The reader is referred to the original article for details. The moral of all this discourse is that rational function integration, while a powerful tool, is an expensive process which is beyond the scope of this project. The methods discussed by Horowitz could possibly be improved by better matrix handling schemes or by other approaches to the problem, e.g. the Risch algorithm mentioned in section 6.3d or doing an integration-by-parts on certain partial fraction forms.

6.3h The heuristics

The heuristics used by the integrator both for deciding which node to expand in a search for solution and within the integration-by-parts routine are devised soley out of personal experience with little regard for proof of their usefulness. The heuristic routines were devised always with the goal in mind of characterising a given integrand to such an extent that a wholly appropriate action might be selected as soon as possible. In situations where two or three routines vie for "most useful," the routine which is most straightforward in logic is applied first. These routines are ones which rely heavily on straightforward algorithms and not so much on heuristics, which generally take time.

It is perhaps possibly that a more efficient IDEA could be written by making a careful cost analysis of all the given options within a particular routine. Such a cost analysis might be based on such things as the number of operators present in an expression's polish tree representation, the types of operators present, size of exponent present, or countless others. The tradeoff between an uncomplicated cost analysis and an efficient heuristic must always be weighed. Just how this weighing might take place is greatly

dependent on the heuristic strategy chosen.

All of the experience used in writing the heuristic routines come from rules which the programmer has learned. An interesting project for further research would then be to write a learning routine for the integrator which could modify its own behaviour on the basis of the experience it has. This would be an awesome project which has never been done entirely successfully. It would require that the coding for the program could be dynamically changed. It would require a long term store to hold the experiences of each integration it tries. If it were implemented as a background program to a time-sharing system, then it could ponder previous integrations in between uses and while the time-sharing system is otherwise idle. The time and resources for such an undertaking are not available to the author at present, hence he made no attempt to implement any such program.

6.3i Improvements to the simplifier

The simplifier, though very adequate for the environment in which it works, could be improved. Rules could be added for special simplifications of specific functions, e.g.

$$\arcsin(\sin x) = x$$

Once such rules are added it is difficult to know where to stop, and the resources available to the author did not allow for all such rules to be implemented. The system would seem lacking if a function simplifier were only partially implemented. Hence, it is entirely omitted.

Other improvements were not implemented because of time or space limitations. A factorisation routine which brought common factors to the outside of the expressions could help the cancellation of fractional exprssions. A reorganisation of the routine into subroutine for each operator

would make the routine more readable. Most simplification needs a reference point. In differentiations with respect to x, a "y" in the expression could be considered constant and thus grouped with other constants. A reference parameter could therefore be added. Interpretation of the reference parameter would depend on the application.

6.4 The legacy of IDEA

The major strategies and the implementation of IDEA, a symbolic integration program, have now been described. Various problems in the project have been met or, at least, discussed. The integration routine is seen to be really an extraordinary simplification routine. The simplification routine described in chapter IV took as its input an algebraic expression in tree form. It looked for certain subtrees, e.g. addition to zero, and changed the tree so that a concise and legible expression tree resulted which retained algebraic equivalence to the inputted expression. The integration routine described in chapter V took as its input an integrand represented by a tree along with a variable of integration also represented in tree form. This routine also looked for certain subtrees, using the variable of integration as a reference, and sought to transform the integrand into some equivalent form this time not for output, but rather which could be found in the integral tables. The transformation could be made directly on the integrand or implicitly by changing the variable of integration. Transformations which removed a substructure work together to reduce the integrand to nothing. Others were checked to make sure that progress towards a solution was being made.

In both the simplification and the integration routines, the form inputted by the user is preserved to as great an extent as possible through-

out the transformations. This ensures that the user receives as output a form which he expects and which is meaningful to him. The assumption is that users who use a symbolic integration program will give intelligent input which contain only constructions which he really meant to have there for some reason or another. By preserving these structures, rather than changing them for computational convenience, it is envisaged that the context and logic in the user's original expression will be preserved.

The recursive definition of a tree as linked trees make it very operable by recursive techniques. This makes trees a very convenient vehicle for representing expressions when the expressions are to be operated on using SASL, a language which makes heavy use of recursion. The use of trees and recursion help make IDEA a very readable symbolic manipulator.

By creating a simplifier based on logic and some good calculus primers, an integrator is created which reads much as does most persons' recollections of a calculus course rather than as very advanced mathematical fireworks. While there are perhaps recently developed methods for general integration, people like Moses and Horowitz show that these methods take up more time and effort than they are worth for all but the most difficult integration problems. For this reason, the more oldfashioned methods are perhaps preferred in cases where they are adequate both because they consume less resources than do the advanced methods and because they are more readable by most persons. While not all the classic integration methods are supplied in IDEA, a very adequate integrator is provided and it is hoped that the previous section sheds some light on how other methods (and which other methods) might be provided.

IDEA does solve a wide variety of problems. So long as the coding remains intact, it does not forget how to do a given integration and thus

remains a useful tool. Furthermore, by creating a program which displays artificial intelligence, insight is gained into human intelligence. IDEA is a record of the intelligent processes of integration and their organisation.

BIBLIOGRAPHY

1. Amarel, S., "On Representations of Problems of Reasoning about Actions," Machine Intelligence, 3, Edinburgh University Press, Edinburgh, 1968.

2. Ayres, Jr., F., Schaum's Outline of Theory and Problems of Differential and Integral Calculus, 2nd ed., McGraw Hill, New York, 1964.

3. Church, A., The Calculi of Lambda Conversion, Princeton University Press, Princeton, New Jersey, 1941.

4. Doran, J., "An Approach to Automatic Problem Solving," Machine Intelligence, 7, Oliver & Boyd, Edinburgh, 1967. See also Doran & Michie, "Experiments with the Graph Traverser Program," Proc. R. Soc. (a), 294, p.235.

5. Doran, J., "New Developments of the Graph Traverser," Machine Intelligence, 2, Oliver & Boyd, Edinburgh, 1968.

6. Feigenbaum, E., and Feldman, J., Computers and Thought, McGraw Hill, London, 1963.

7. Fisher, R. C., and Ziebur, A. D., Calculus and Analytic Geometry, 2nd ed., Prentice-Hall, Englewood Cliffs, New Jersey, 1965.

8. Fogel, L. J., Owens, A. J., and Walsh, M. J., Artificial Intelligence through Simulated Evolution, Wiley & Sons, London, 1966.

9. Gries, D., Compiler Construction for Digital Computers, Wiley & Sons, London, 1971.

10. Hardy, G. H., The Integration of Functions of a Single Variable, Cambridge University Press, Cambridge, 1905.

11. Henrici, P., Elements of Numerical Analysis, Wiley & Sons, London, 1964.

12. Horowitz, E., "Algorithms for Partial Fraction Decomposition and Rational Function Integration," Proc. 2nd Symposium on Symbolic and Algebraic Manipulation, ACM, 1971.

13. Hunt, E. B., Artificial Intelligence, Academic Press, London, 1975.

14. Knuth, D. E., The Art of Computer Programming, Addison-Wesley, Reading, Massachusetts.

15. Manove, M., Bloom, S., and Engelman, C., "Rational Functions in MATHLAB" Symbol Manipulation Languages and Techniques, North-Holland Publishing Co., Amsterdam, 1968.

16. McCarthy, J., Lisp 1.5 Programmer's Manual, MIT Press, Cambridge, Massachusetts, 1965.

17. Michie, D., and Ross, R., "Experiments with the Adaptive Graph Traverser," Machine Intelligence, 5, Edinburgh University Press, Edinburgh, 1969.

18. Michie, D., "Strategy Building with the Graph Traverser," Machine Intelligence, 7, Oliver & Boyd, Edinburgh, 1967.

19. Moses, J., "The Evolution of Algebraic Manipulation Algorithms," Proc. IFIP Congress 74, 3, p.483.

20. Moses, J., "Algebraic Simplification: A guide for the Perplexed," Comm. ACM, 14 no. 8 Aug. 1971, p.527.

21. Moses, J., "Symbolic Integration: The Stormy Decade," Comm. ACM, 14, no. 8, Aug, 1971, p.548.

22. Moses, J., Symbolic Integration, Project MAC, TR-47, Massachusetts Institute of Technology, 1967.

23. Nilsson, N. J., Problem-solving Methods in Artificial Intelligence, McGraw-Hill, London, 1971.

24. Pohl, I., "First Results on the Effect of Error in Heuristic Search," Machine Intelligence, 4, Edinburgh University Press, Edinburgh, 1969.

25. Pohl, I., "Heuristic Search Viewed as Path Finding in a Graph," Art. Intell., 1, 1970, p.193.

26. Risch, R., "The Problem of Integration in Finite Terms," Trans. AMS, 139, May 1969, p.167.

27. Slagle, J.R., A Heuristic Program that Solves Symbolic Integration in Freshman Calculus, Ph. D. Diss., Massachusetts Institute of Technology, 1961. (See also Feigenbaum and Feldman)

28. Slagle, J. R., Artificial Intelligence: The Heuristic Programming Approach, McGraw-Hill, London, 1971.

29. Turing, A. M., "Computing Machinery and Intelligence," Mind, 59, (n. s. 236), p.433. (See also Feigenbaum and Feldman)

30. Turner, D. A., SASL Language Manual, Document CS/75/1, revised 16/9/75, St. Andrews University Computing Laboratory, St. Andrews.

31. Weissman, C., Lisp 1.5 Primer, Dickenson Publishing Company, Belmont California, 1967.

APPENDIX A

PROGRAM LISTING

```
||
||
||      GLOBALLY NEEDED FUNCTIONS
||
||

@NOLIST  || DISLPR      THE SASL 'PRELUDE'
|| LAYOUT FUNCTIONS
_LET NWIDTH X    || WIDTH OF NUMBER X WHEN PRINTED
_BE X<0 -> 1+NWIDTH(-X); X<10-> 1; 1+NWIDTH(X/10)
_AND WIDTH X     || WIDTH OF OBJECT X WHEN PRINTED
_BE _NUMBER X->NWIDTH X;
    _CHAR Y->1;
    X=_TRUE->5;X=_FALSE->6;
    _FUNCTION X->10;
    X=()->0; WIDTH(_HD X)+WIDTH(_TL X)
_LET SPACES N    || A LIST OF N SPACES
_BE N<=0 -> (); (' '),SPACES(N-1)
_NEW LJUSTIFY N X =  X,SPACES(N-WIDTH X)
_NEW RJUSTIFY N X =  SPACES(N-WIDTH X),X,
_NEW CJUSTIFY N X
_BE _NEW S = N-WIDTH X
    _NEW L = S/2 _IN
    SPACES L,X,SPACES(S-L)
_LET STRING X =
  X=() -> _TRUE;
  _LIST X -> _CHAR(_HD X) -> STRING (_TL X);_FALSE;
  _FALSE
_NEW STRING X = X=() -> _FALSE;STRING X
_LET DISPLAY X  || MAKES THE STRUCTURE OF X VISIBLE
_BE X=_NL -> '_NL";
    _CHAR X -> '%',X,;
    STRING X -> '"', X , '"', ;
    _LIST X -> '(',DISPLAYLIST X,')',; X
_AND DISPLAYLIST X
_BE X=() -> ();
DISPLAY(_HD X),',',DISPLAYLIST(_TL X)
_NEW DISPLAY X _BE (X=()|~_LIST X|STRING X -> DISPLAY;DISPLAYLIST)X
||DISPLAY X MAKES THE STRUCTURE OF X VISIBLE ON OUTPUT
_LET LENGTH L      || THE LENGTH OF LIST L
_BE L=() -> 0;1 + LENGTH(_TL L)
_LET SHUNT X Y
_BE X=() -> Y; SHUNT(_TL X)(_HD X,Y)
_NEW REVERSE X _BE SHUNT X()
```

```
_LET APPEND X Y    || CONCATENATE THE LISTS X AND Y
_BE SHUNT(REVERSE X) Y
_LET FOR A B F     || RETURNS THE LIST F A,F(A+1), ...,F B,
_BE A>B  -> (); F A,FOR(A+1) B F
_LET MAP F X       || 'MAPS' F ALONG THE LIST X
_BE X=() -> (); F(_HD X),MAP F(_TL X)
_LET COUNT A B     || THE LIST A,A+1, ...,B,
_BE A>B -> (); A,COUNT(A+1) B
_LET SELECT N X    || THE N'TH COMPONENT OF X
_BE N=1 -> _HD X; SELECT(N-1)(_TL X)
_LET ZIP X
_BE _NEW N=LENGTH(_HD X)
    _NEW F I _BE MAP(SELECT I) X
    _IN FOR 1 N F
_LET WHILE F G X || 'BEGIN WHILE F(X) DO X:= G(X); X END'(ALGOL EQUIVALENT)
_BE F X -> WHILE F G(G X);
    X
_AND UNTIL F G X  || 'WHILE NOT'
_BE F X -> X;
    UNTIL F G(G X)
|| CURRY'S COMBINATORS
_LET S F G X = F X (G X)
_AND K X Y = X
_AND Y H = _LET F = H F _IN F
_AND C F G X = F X G
_AND B F G X = F(G X)
_AND I X = X
_AND W F X = F X X
_AND KI X Y = Y
_IN ?LIST
?NOLIST
_NEW SIN,COS,TAN,COT,SEC,CSC,='SIN",'COS",'TAN",'COT",'SEC",'CSC",
_LET ATOMIC X _BE ¬_LIST X | STRING X
_LET POS X ||  SEES IF X IS POSITIVE
_BE _NUMBER X -> X¬=0; ¬(_HD X=-_)
_LET SUBST P Q R  ||  SUBSTITUTE P FOR Q IN R
_BE R=() -> ();
    R=Q -> P;
    ATOMIC R -> R=Q -> P; R;
    SUBST P Q (_HD R) ,  SUBST P Q (_TL R)
_IN

_LET NOTCONTAINS X S     ||  IS X NOT PRESENT IN S??
_BE S=X -> _FALSE;
```

```
    ATOMIC S -> _TRUE;
    S=() -> _TRUE;
    _LET (H,T)=S
    _IN NOTCONTAINS X H  &  NOTCONTAINS X T


_LET DIFF S VAR   _BE
  S=() -> ();
  S=VAR -> 1 ;
  ATOMIC S -> 0 ;
  DIFFOP (_HD S) VAR (_TL S)
_AND DIFFOP S VAR   _BE
  S="_ -> _LAMBDA(OPERAND,).
    "_,DIFF OPERAND VAR,;
  S="+ |  S="- -> _LAMBDA(LEFT,RIGHT,).
    S,DIFF LEFT VAR,DIFF RIGHT VAR,;
  S="* -> _LAMBDA(LEFT,RIGHT,).
    "+,("*,LEFT,DIFF RIGHT VAR,),("*,RIGHT,DIFF LEFT VAR,),;
  S="/ -> _LAMBDA(LEFT,RIGHT,).
    "/,("-,("*,RIGHT,DIFF LEFT VAR,),("*,LEFT,DIFF RIGHT VAR,),),
              ("*,RIGHT,RIGHT,),;
  S= SIN  -> _LAMBDA(OPERAND,).
  "*,( COS ,OPERAND,), DIFF OPERAND VAR ,;
  S='EXPN" -> _LAMBDA(OPERAND,).
      "*,('EXPN",OPERAND,), DIFF OPERAND VAR ,;
  S= COS  -> _LAMBDA(OPERAND,).
      "_,("*,( SIN ,OPERAND,),   DIFF OPERAND VAR ,),;
  S= TAN  -> _LAMBDA(OPERAND,).
      "*,('**",( SEC ,OPERAND,),2,), DIFF OPERAND VAR ,;
  S='**" -> _LAMBDA(LEFT,RIGHT,).
   NOTCONTAINS VAR RIGHT
      ->          "*,("*,RIGHT,('**",LEFT,SIMPLIFY("-,RIGHT,1,),),),
        DIFF LEFT VAR,;
  "*,('**",LEFT,RIGHT,),("+,("*,(%/,RIGHT,LEFT,),DIFF LEFT VAR,),
            ("*,('LN",LEFT ,),DIFF RIGHT VAR ,),),;
 S= SEC  -> _LAMBDA(OPERAND,).
        "*,("*,( SEC ,OPERAND,),( TAN ,OPERAND,),),DIFF OPERAND VAR,;
 S= CSC  -> _LAMBDA(OPERAND,).
        "_,("*,("*,( CSC ,OPERAND,),( COT ,OPERAND,),),
            DIFF OPERAND VAR,),;
 S= COT  -> _LAMBDA(OPERAND,).
     "_,("*,('**",( CSC ,OPERAND,),2,),DIFF OPERAND VAR,),;
 S='ARCSIN" -> _LAMBDA(OPERAND,).
```

```
                    DIFF OPERAND VAR,;
S='ARCCOS" -> _LAMBDA(OPERAND,).
     ~_,(~*,(~/,1,('**",(~-,1,('**",OPERAND,2,),),(~/,1,2,),),),
                    DIFF OPERAND VAR,),;
S='ARCTAN" -> _LAMBDA(OPERAND,).
          ~*,(~/,1,(~+,1,('**",OPERAND,2,),),),DIFF OPERAND VAR,;
S='ARCCOT" -> _LAMBDA(OPERAND,).
 ~_,(~*,(~/,1,(~+,1,('**",OPERAND,2,),),),DIFF OPERAND VAR,),;
S='ARCSEC" -> _LAMBDA(OPERAND,).
   ~*,(~/,1,(~*,OPERAND,('**",(~-,('**",OPERAND,2,),1,),
       (~/,1,2,),),),),DIFF OPERAND VAR,;
S='ARCCSC" -> _LAMBDA(OPERAND,).
     ~_,(~*,(~/,1,(~*,OPERAND,('**",(~-,('**",OPERAND,2,),1,),
          (~/,1,2,),),),),DIFF OPERAND VAR,),;
S='SINH"-> _LAMBDA(OPERAND,).
   ~*,('COSH",OPERAND,),DIFF OPERAND VAR,;
S='COSH" -> _LAMBDA(OPERAND,).
          ~*,('SINH",OPERAND,),DIFF OPERAND VAR,;
S='TANH" -> _LAMBDA(OPERAND,).
     ~*,('**",('SECH",OPERAND,),2,),DIFF OPERAND VAR,;
S='COTH" -> _LAMBDA(OPERAND,).
     ~_,(~*,('**",('CSCH",OPERAND,),2,),DIFF OPERAND VAR,),;
S='SECH" -> _LAMBDA(OPERAND,).
     ~_,(~*,(~*,('SECH",OPERAND,),),('TANH",OPERAND,),
               DIFF OPERAND VAR,),;
S='CSCH" -> _LAMBDA(OPERAND,).
     ~_,(~*,(~*,('CSCH",OPERAND,),('TANH",OPERAND,),),
                DIFF OPERAND VAR,),;
S='ARCSINH" -> _LAMBDA(OPERAND,).
     ~*,(~/,1,('**",(~-,1,('**",OPERAND,2,),),  (~/,1,2,),),),
                  DIFF OPERAND VAR,;
S='ARCCOSH" -> _LAMBDA(OPERAND,).
     ~*,(~/,1,('**",(~-,('**",OPERAND,2,),1,),(~/,1,2,),),),
                DIFF OPERAND VAR,;
S='ARCTANH" | S='ARCCOTH" -> _LAMBDA(OPERAND,).
     ~*,(~/,1,(~-,1,('**",OPERAND,2,),),),DIFF OPERAND VAR,;
S='ARCSECH" -> _LAMBDA(OPERAND,).
     ~_,(~/,1,(~*,OPERAND,('**",(~-,1,('**",OPERAND,2,),),
          (~/,1,2,),),),  DIFF OPERAND VAR,),;
S='ARCCSCH" -> _LAMBDA(OPERAND,).
     ~_,(~*,(~/,1,(~*,OPERAND,('**",(~+,1,('**",OPERAND,2,),),
          (~/,1,2,),),),),      DIFF OPERAND VAR,),;
S='LN" -> _LAMBDA(OPERAND,).
     ~*,(~/,1,OPERAND,),DIFF OPERAND VAR,;
```

```
177          _LAMBDA(OPERAND,).
178             %*,((APPEND S (%',)),OPERAND,),DIFF OPERAND VAR,
179
180          ||
181          ||
182          ||       SIMPLIFICATION ROUTINES
183          ||
184          ||
185          _AND GATHER TERMS  ||  GET RID OF PLUS OR TIMES SIGNS .
186          _BE _NEW OP=_HD TERMS
187              _LET GATHER (A,B,)
188              _BE APPEND (FIX A)(FIX B)
189             _AND FIX X
190              _BE _LIST X -> _HD X=OP -> GATHER(_TL X);
191                             X,;
192                 X,
193              _IN GATHER (_TL TERMS)
194
195          _AND EQUAL X Y  ||  EQUALITY CONSIDERING COMMUTATIVE PROPERTY
196          _BE X=Y -> _TRUE;
197              ATOMIC X | ATOMIC Y -> _FALSE;
198              _TL _TL X=() & _TL _TL Y=() -> EQUAL (_HD X)(_HD Y) &
199                                             EQUAL (_HD _TL X)(_HD _TL Y);
200              _HD X=%+ | _HD X=%*
201              -> _LET TESTMEM X Y
202                 _BE X=() -> Y=();
203                     Y=() -> _FALSE;
204                     _NEW A,T=MEMBER(_HD X) Y
205                     _IN A -> TESTMEM(_TL X) T;
206                              _FALSE
207                 _IN _HD X=_HD Y -> TESTMEM (GATHER X)(GATHER Y);
208                     _FALSE;
209              _FALSE
210
211          _AND MEMBER X L
212          _BE L=()  -> _FALSE,;
213              EQUAL X (_HD L) -> _TRUE,_TL L; || CROSS THAT MEMBER OFF
214              _LET A,T=MEMBER X (_TL L)
215              _IN A -> A,_HD L,T;
216                  A,T
217
218          _AND SIMPLIFY S  ||  ALGEBRAIC SIMPLIFICATION
219          _BE S=() ->();
220              _NUMBER S -> S<0 ->%_,-S,;
```

```
ATOMIC -> S;
_NEW SIMPL=SIMPLIFY(_HD _TL S)   || NOW LOOKING FOR MONADIC FUNCTION
_IN _HD S=%_ -> SIMPL=0 -> 0;
                ATOMIC SIMPL -> S;
                _HD SIMPL=%_ -> _HD _TL SIMPL;
                %_,SIMPL,;
     _TL _TL S=() -> _HD S,SIMPL,; || MONADIC FUNCTION FOUND
     _NEW SIMPR=SIMPLIFY(_HD _TL _TL S)
     _IN _HD S='**"    || SIMPLIFY EXPONENTIATIONS
         -> SIMPR=0 -> 1;
            SIMPL=1 -> 1;
            SIMPR=1 -> SIMPL;
            _NUMBER SIMPL & _NUMBER SIMPR -> POWER SIMPL SIMPR;
            ATOMIC SIMPL -> '**",SIMPL,SIMPR,;
            _HD SIMPL='**" -> SIMPLIFY('**",_HD _TL SIMPL,
                                        (%*,_HD _TL _TL SIMPL,
                                            SIMPR,),);

             _HD SIMPL=%_
             -> _NUMBER SIMPR
                -> SIMPR _MOD 2=1 -> SIMPLIFY(%_,('**",_HD _TL SIMPL,
                                                  SIMPR,),);

                   SIMPLIFY('**",_HD _TL SIMPL,SIMPR,);
               '**",SIMPL,SIMPR,;
             _HD SIMPL=%/| _HD SIMPL=%*
             -> _NEW B,Y,T,=SIMPL
                _IN SIMPLIFY(B,('**",Y,SIMPR,),('**",T,SIMPR,),);
             _HD SIMPL='EXPN" -> 'EXPN",SIMPLIFY(%*,SIMPR,_TL SIMPL),;
             '**",SIMPL,SIMPR,;
          _HD S=%-    ||   SIMPLIFY SUBTRACTIONS
          -> _NUMBER SIMPR
             -> _NUMBER SIMPL -> SIMPLIFY(SIMPL-SIMPR);
                SIMPR=0 -> SIMPL;
                _HD SIMPL=%/ -> SIMPLIFY (COMDENOM S);
             %-,SIMPL,SIMPR,;
             _NUMBER SIMPL -> SIMPL=0 -> %_,SIMPR,;
                              _HD SIMPR=%/ -> SIMPLIFY(COMDENOM S);
                             %-,SIMPL,SIMPR,;
             EQUAL SIMPL SIMPR -> 0;
             _HD SIMPR=%/ | _HD SIMPL=%/ -> SIMPLIFY(COMDENOM S);
             _HD SIMPR=%_
             -> _HD SIMPL=%_ -> SIMPLIFY(%-,_HD _TL SIMPR,_HD _TL SIMPL,);
                SIMPLIFY(%+,SIMPL,_HD _TL SIMPR,);
             _HD SIMPR=%*
```

```
265                     ->EQUAL(_HD _TL _TL SIMPR) SIMPL
266                       -> SIMPLIFY(%*,(%-,1,_HD _TL SIMPR,),SIMPL,);
267                       _HD SIMPL=%*
268                       -> EQUAL(_HD _TL _TL SIMPR)(_HD _TL _TL SIMPL)
269                          -> SIMPLIFY(%*,(%-,_HD _TL SIMPL,_HD _TL SIMPR,),
270                                        _HD _TL _TL SIMPL,);
271                          %-,SIMPL,SIMPR,;
272                       %-,SIMPL,SIMPR,;
273                     _HD SIMPL=%* -> EQUAL SIMPR (_HD _TL _TL SIMPL)
274                                     -> SIMPLIFY(%*,(%-,_HD _TL SIMPL,1,),SIMPR,);
275                                     %-,SIMPL,SIMPR,;
276                     ATOMIC SIMPL
277                     -> _HD SIMPR=%+
278                      ->_NEW((SIMPL,),SIMPR)=SUBTRACT(SIMPL,)(GATHER SIMPR)
279                        _IN SIMPR=() -> SIMPL;
280                            %-,SIMPL,SIMPLIFY(ADDPOLY SIMPR),;
281                      %-,SIMPL,SIMPR,;
282                     _HD   SIMPL=%+
283                       -> _NEW (SIMPL,SIMPR)
284                          = ATOMIC SIMPR -> SUBTRACT(GATHER SIMPL)(SIMPR,);
285                            _HD SIMPR=%+ -> SUBTRACT(GATHER SIMPL)(GATHER SIMPR);
286                            SUBTRACT (GATHER SIMPL)(SIMPR,)
287                          _IN SIMPR=() -> SIMPLIFY(ADDPOLY SIMPL);
288                              %-,SIMPLIFY(ADDPOLY SIMPL),SIMPLIFY(ADDPOLY SIMPR),;
289                     _HD SIMPR=%+
290                     -> _NEW (SIMPL,SIMPR)=SUBTRACT(SIMPL,)(GATHER SIMPR)
291                        _IN SIMPR=() -> SIMPLIFY(ADDPOLY SIMPL);
292                            %-,SIMPLIFY(ADDPOLY SIMPL),SIMPLIFY(ADDPOLY SIMPR),;
293                   %-,SIMPL,SIMPR,;
294                     _HD S=%/   ||   SIMPLIFY DIVISIONS
295                     -> SIMPR=1 -> SIMPL;
296                        SIMPL=0 -> 0;
297                        _NUMBER SIMPL & _NUMBER SIMPR
298                        -> _NEW A=GCD SIMPL SIMPR
299                           _IN SIMPR/A=1 -> SIMPL/A;
300                               %/,SIMPL/A,SIMPR/A,;
301                        EQUAL SIMPL SIMPR -> 1;
302                        ATOMIC SIMPL -> ATOMIC SIMPR -> %/,SIMPL,SIMPR,;
303                                        _HD SIMPR=%_ -> SIMPLIFY(%_,(%/,SIMPL,
304                                                                     _HD _TL
305                                                                     SIMPR,),);
306                                        _HD SIMPR='**"
307                                        -> EQUAL SIMPL(_HD _TL SIMPR)
308                                           -> SIMPLIFY('**",SIMPL,
```

```
                                                    (%-,1,_HD _TL _TL SIMPR,),);
                                        %/,SIMPL,SIMPR,;
                                    _HD SIMPR=%* -> CANCEL SIMPL SIMPR;
                                    _HD SIMPR=%/
                                    -> SIMPLIFY(%/,(%*,_HD _TL _TL SIMPR,SIMPL,),
                                                   _HD _TL SIMPR,);
                                    %/,SIMPL,SIMPR,;
                        _HD SIMPL=%_ ->SIMPLIFY(%_,(%/,_HD _TL SIMPL,SIMPR,),);
                        _HD SIMPL='**"      -> EQUAL SIMPR (_HD _TL SIMPL)
                                               -> SIMPLIFY('**",SIMPR,
                                                           (%-,_HD _TL _TL SIMPL,1,
                                                           ),);
                                            ATOMIC SIMPR -> %/,SIMPL,SIMPR,;
                                            _HD SIMPR='**"
                                            -> EQUAL (_HD _TL SIMPL) (_HD _TL SIMPR)
                                               -> '**",_HD _TL SIMPL,
                                                   SIMPLIFY(%-,_HD _TL _TL SIMPL,
                                                            _HD _TL _TL SIMPR,),;
                                               %/,SIMPL,SIMPR,;
                                            %/,SIMPL,SIMPR,;
                        _HD SIMPL=%/ -> SIMPLIFY(%/,_HD _TL SIMPL,
                                                 (%*,_HD _TL _TL SIMPL,SIMPR,),);
                        _HD SIMPL=%* -> ATOMIC SIMPR -> CANCEL SIMPL SIMPR;
                                        _HD SIMPR=%/
                                        -> SIMPLIFY(%/,(%*,_HD _TL _TL SIMPR,SIMPL,),
                                                       _HD _TL SIMPR,);
                                        CANCEL SIMPL SIMPR;
                         ATOMIC SIMPR -> %/,SIMPL,SIMPR,;
                         _HD SIMPR='**" -> EQUAL (_HD _TL SIMPR) SIMPL
                                           -> SIMPLIFY('**",SIMPL,
                                                       (%-,1,_HD _TL _TL SIMPR,),);
                                        %/,SIMPL,SIMPR,;
                          _HD SIMPR=%_ -> SIMPLIFY(%_,(%/,SIMPL,_HD _TL SIMPR,),);
_HD SIMPR=%/ -> SIMPLIFY(%/,(%*,_HD _TL _TL SIMPR,SIMPL,),
                             _HD _TL SIMPR,);
                         _HD SIMPR=%* | _HD SIMPL=%* -> CANCEL SIMPL SIMPR;
                         %/,SIMPL,SIMPR,;
                    _HD S=%*  ||  SIMPLIFY MULTIPLICATIONS
                    -> _NUMBER SIMPR -> _NUMBER SIMPL -> SIMPL*SIMPR;
                                        SIMPR=0 -> 0;
                                        SIMPR=1 -> SIMPL;
                                        COLLECT(%*,SIMPR,SIMPL,);
                         _NUMBER SIMPL -> SIMPL=0 -> 0;
                                        SIMPL=1 -> SIMPR;
```

```
                                COLLECT(%*,SIMPL,SIMPR,);
                        EQUAL SIMPL SIMPR-> SIMPLIFY('**",SIMPL,2,);
                        _HD SIMPL=%_
                        -> _HD SIMPR=%_ -> SIMPLIFY(%*,_HD _TL SIMPL,
                                                    _HD _TL SIMPR,);
                           SIMPLIFY(%_,(%*,_HD _TL SIMPL,SIMPR,),);
                        _HD SIMPR=%_ -> SIMPLIFY(%_,(%*,SIMPL,
                                                    _HD _TL SIMPR,),);
                        _HD SIMPL=%/
                        -> _HD SIMPR=%/
                           -> SIMPLIFY(%/,(%*,_HD _TL SIMPL,_HD _TL SIMPR,),
                                          (%*,_HD _TL _TL SIMPL,_HD _TL _TL SIMPR,
                                           ),);
                            SIMPLIFY(%/,(%*,_HD _TL SIMPL,SIMPR,),
                                          _HD _TL _TL SIMPL,);
                         _HD SIMPR=%/
                         -> SIMPLIFY(%/,(%*,_HD _TL SIMPR,SIMPL,),
                         _HD _TL _TL SIMPR,);
                        _HD SIMPR='**"
                        -> EQUAL SIMPL (_HD _TL SIMPR)
                           -> SIMPLIFY('**",SIMPL,(%+,_HD _TL _TL SIMPR,1,),);
                           _HD SIMPL='**"
                           -> EQUAL(_HD _TL SIMPL)(_HD _TL SIMPR)
                              -> SIMPLIFY('**",_HD _TL SIMPL,
                                          (%+,_HD _TL _TL SIMPL,
                                              _HD _TL _TL SIMPR,),);
                              %*,SIMPL,SIMPR,;
                           %*,SIMPL,SIMPR,;
                           _HD SIMPL='**" -> SIMPLIFY(%*,SIMPR,SIMPL,);
                     COLLECT(%*,SIMPL,SIMPR,);
_HD S=%+     ||  SIMPLIFY ADDITIONS
-> _NUMBER SIMPR -> _NUMBER SIMPL -> SIMPL+SIMPR;
                        SIMPR=0 -> SIMPL;
                        COLLECT(%+,SIMPR,SIMPL,);
                     _NUMBER SIMPL ->SIMPL=0 -> SIMPR;
                        COLLECT(%+,SIMPL,SIMPR,);
                     EQUAL SIMPL SIMPR -> COLLECT(%*,2,SIMPL,);
                     _HD SIMPR=%/ & _HD SIMPL=%/
                     -> EQUAL(_HD _TL _TL SIMPL)(_HD _TL _TL SIMPR)
                        ->SIMPLIFY(%/,(%+,_HD _TL SIMPL,_HD _TL SIMPR,),
                                       _HD _TL _TL SIMPL,);
                        SIMPLIFY(COMDENOM (%+,SIMPL,SIMPR,));
                     _HD SIMPR=%/  | _HD SIMPL=%/ -> SIMPLIFY(COMDENOM
                                                              (%; SIMPL
```

```
                                                                ,SIMPR,));
                  _HD SIMPR=%_
                  -> _HD SIMPL=%_ -> %_,COLLECT(%+,_HD _TL SIMPL,
                                                   _HD _TL SIMPR,),;
                     EQUAL SIMPL(_HD _TL SIMPR) -> 0;
                     SIMPLIFY(%-,SIMPL,_HD _TL SIMPR,);
                  _HD SIMPL=%_
                  -> EQUAL SIMPR (_HD _TL SIMPL) -> 0;
                  SIMPLIFY(%-,SIMPR,_HD _TL SIMPL,);
                  COLLECT(%+,SIMPL,SIMPR,);
               S


_AND COLLECT (OP,SIMPL,SIMPR,)  || COMBINATION RULES FOR ADDS AND MULTS
_BE OP=%+     || COMBINE ADDITIONS
    -> _HD SIMPR=%+
       ->_NUMBER _HD _TL SIMPR
         ->_NUMBER SIMPL -> COLLECT(%+,SIMPL+_HD _TL SIMPR,
                                       _HD _TL _TL SIMPR,);
           _HD SIMPL=%+
           -> _NUMBER _HD _TL SIMPL
              ->COLLECT(%+,_HD _TL SIMPL+_HD _TL SIMPR,
                           (%+,_HD _TL _TL SIMPL,_HD _TL _TL SIMPR,),);
              %+,_HD _TL SIMPR,(%+,SIMPL,_HD _TL _TL SIMPR,),;
            %+,SIMPL,SIMPR,;
          %+,SIMPL,SIMPR,;
         _HD SIMPR=%/ -> SIMPLIFY(COMDENOM(OP,SIMPL,SIMPR,));
         EQUAL (SIMPLIFY(%_,SIMPL,)) SIMPR -> 0;
         _NUMBER SIMPL -> OP,SIMPL,SIMPR,;
         _HD SIMPL=%/ -> SIMPLIFY(COMDENOM(OP,SIMPL,SIMPR,));
         _HD SIMPR=%*
         ->EQUAL SIMPL (_HD _TL _TL SIMPR)
           -> SIMPLIFY(%*,(%+,_HD _TL SIMPR,1,),SIMPL,);
           ATOMIC SIMPL -> %*,SIMPL,SIMPR,;
           _HD SIMPL=%*
           -> EQUAL(_HD _TL _TL SIMPL)(_HD _TL _TL SIMPR)
              ->SIMPLIFY(%*,(%+,_HD _TL SIMPL,_HD _TL SIMPR,),
                 _HD _TL_TL SIMPL,);
              %+,SIMPL,SIMPR,;
            %+,SIMPL,SIMPR,;
          %+,SIMPL,SIMPR,;
    OP=%*
    -> _HD SIMPR=%_ -> SIMPLIFY(%_,(OP,SIMPL,_HD _TL SIMPR,),);
       _HD SIMPR=%+
```

```
441                -> _NUMBER SIMPL-> OP,SIMPL,SIMPR,;
442                   _HD SIMPL=%+
443                   -> SIMPLIFY(%+,(%*,_HD _TL SIMPL,_HD _TL SIMPR,),
444                                  (%+,(%*,_HD _TL SIMPL,_HD _TL _TL SIMPR,),
445                                      (%*,_HD _TL _TL SIMPL,_HD _TL _TL SIMPR,
446                                       ),),);
447                     OP,SIMPL,SIMPR,;
448                    _HD SIMPR=%/ -> SIMPLIFY(%/,(%*,SIMPL,_HD _TL SIMPR,),
449                                                _HD _TL _TL SIMPR,);
450                _HD SIMPR=%*
451                ->_NUMBER _HD _TL SIMPR
452                  ->_NUMBER SIMPL
453                    -> COLLECT(%*,SIMPL*_HD _TL SIMPR,_HD _TL _TL SIMPR,);
454                    _HD SIMPL=%*
455                    -> _NUMBER _HD _TL SIMPL -> COLLECT(%*,
456                                                        _HD _TL SIMPL*_HD _TL SIMPR,
457                                                        (%*,_HD_TL_TL SIMPL,
458                                                            _HD_TL_TL SIMPR,),);
459                       %*,_HD _TL SIMPR,(%*,SIMPL,_HD _TL _TL SIMPR,),;
460                   OP,SIMPL,SIMPR,;
461                   ¬ATOMIC SIMPL -> _HD SIMPL=%/
462                                    -> SIMPLIFY(%/,(%*,SIMPR,_HD _TL SIMPL,),
463                                                   _HD _TL _TL SIMPL,);
464                                    OP,SIMPL,SIMPR,;
465                                  OP,SIMPL,SIMPR,;
466                    OP,SIMPL,SIMPR,;
467             OP,SIMPL,SIMPR,
468
469          _AND GCD X Y || EUCLID'S ALGORITHM
470          _BE _NEW A=X _MOD Y
471              _IN A=0 -> Y; GCD Y A
472
473          _AND TIMESPOLY (A,B)   ||  PUT BACK TIMES SIGNS 'GATHER' REMOVED
474          _BE B=() -> A;
475              %*,A,TIMESPOLY B,
476
477          _AND CANCEL L M      || CANCEL THE FRACTION L/M
478          _BE _NEW L=ATOMIC L-> L,; _HD L=%* -> GATHER L; L,
479              _AND M=ATOMIC M -> M,; _HD M=%* ->  GATHER M; M,
480              _LET CAN L M
481              _BE L=() -> (),M;
482                  _NEW (H,T)=L
483                  _NEW(H,M)=CAN1 H M
484                  _NEW (L,M)=CAN T M
```

```
            _IN (H,L),M
        _AND CAN1 A M
        _BE M=() -> A,;
            _NEW (H,T)=M
            _NEW X=SIMPLIFY(%/,A,H,)
            _IN ATOMIC X -> CAN1 X T;
                _HD X¬=%/ -> CAN1 X T;
                _NEW (OP,A,H,)=X
                _NEW (A,M)=CAN1 A T
                _IN A,(H,M)
        _NEW A,B=CAN L M
        _IN B=() -> SIMPLIFY(TIMESPOLY A);
            %/,SIMPLIFY(TIMESPOLY A),SIMPLIFY(TIMESPOLY B),

_AND POWER X Y      ||  RAISE X TO THE Y POWER
_BE Y=0 -> 1; X*POWER X (Y-1)

_AND COMDENOM S ||FIND A COMMON DENOMINATOR
_BE ATOMIC S -> S; _TL _TL S=() -> S;
    _NEW OP,LEFT,RIGHT,=S
    _NEW LEFT=ATOMIC LEFT -> LEFT;
              _HD LEFT=%+ | _HD LEFT=%- -> COMDENOM LEFT;
              LEFT
    _AND RIGHT=ATOMIC RIGHT -> RIGHT;
               _HD RIGHT=%+ | _HD RIGHT=%- -> COMDENOM RIGHT;
               RIGHT
    _IN OP='**" -> S;
        ATOMIC RIGHT
        -> ATOMIC LEFT -> OP,LEFT,RIGHT,;
           _HD LEFT ¬= %/ -> OP,LEFT,RIGHT,;
           (%/,(OP,_HD _TL LEFT,(%*,RIGHT,_HD _TL _TL LEFT,),),
                _HD _TL _TL LEFT,);
        ATOMIC LEFT -> _HD RIGHT¬=%/ -> OP,LEFT,RIGHT,;
                       (%/,(OP,(%*,_HD _TL _TL RIGHT,LEFT,),_HD _TL RIGHT,),
                           _HD _TL _TL RIGHT,);
        _HD RIGHT=%/
        -> _HD LEFT=%/
           -> EQUAL(_HD _TL _TL LEFT)(_HD _TL _TL RIGHT)
              ->%/,(OP,_HD _TL LEFT,_HD _TL RIGHT,),_HD _TL _TL LEFT,;
              (%/,(OP,(%*,_HD _TL _TL RIGHT,_HD _TL LEFT,),
                      (%*,_HD _TL _TL LEFT,_HD _TL RIGHT,),),
                  (%*,_HD _TL _TL LEFT,_HD _TL _TL RIGHT,),);
           (%/,(OP,(%*,_HD _TL _TL RIGHT,LEFT,),_HD _TL RIGHT,),
                   _HD _TL _TL RIGHT,);
```

```
529             _HD LEFT=%/ -> %/,(OP,_HD _TL LEFT,(%*,_HD _TL _TL LEFT,RIGHT,),),
530                                 _HD _TL _TL LEFT,;
531              OP,LEFT,RIGHT,
532
533     _AND SUBTRACT L M       || FIND HIDDEN SUBTRACTIONS
534     _BE L=() -> (),M;
535         _NEW (H,T)=L
536         _NEW (H,M)=SUB1 H M
537         _NEW (L,M)=SUBTRACT T M
538         _IN (H,L),M
539     _AND SUB1 A M
540     _BE M=() -> A,;
541         _NEW(H,T)=M
542         _NEW X=SIMPLIFY(%-,A,H,)
543         _IN ATOMIC X -> SUB1 X T;
544             _HD X¬=%- -> SUB1 X T;
545             _NEW (OP,A,H,)=X
546             _NEW (A,M)=SUB1 A T
547             _IN A,(H,M)
548     _AND ADDPOLY(A,B)   || MAKE LIST (A,B) INTO POLISH ADDITION
549     _BE B=() -> A;
550         %+,A,ADDPOLY B,
551
552
553     ||
554     ||
555     ||        AUXILARY ROUTINES FOR MAIN INTEGRATION ROUTINES
556     ||
557     ||
558
559     _NEW FAILURE=_FALSE,(),(),
560     _LET POLYNOMIAL S VAR ||  IS S A POLYNOMIAL IN VAR???
561     _BE  ATOMIC S -> S=VAR -> _TRUE,1,(1,0,),; _TRUE,0,(S,),;
562          S=VAR -> _TRUE,1,(1,0,),;
563         _HD S=%/ -> SIMPLIFY(DIFF S VAR)=0 -> _TRUE,0,(S,),; FAILURE;
564          _HD S=%+ | _HD S=%- ->
565           _NEW A,B,C,=POLYNOMIAL(_HD _TL S)VAR
566           _AND X,Y,Z,=_HD S=%+ -> POLYNOMIAL(_HD _TL _TL S)VAR;
567                                   NEGATE(POLYNOMIAL(_HD _TL _TL S)VAR)
568           _IN ¬A | ¬X ->  FAILURE;
569               B>=Y -> _TRUE,B,ADD C Z,; _TRUE,Y,ADD Z C,;
570                     ||  RETURNS DEGREE AND COEFFICIENTS
571         _HD S=%* ->
572           _NEW P,Q,=_TL S
```

```
573             _IN SIMPLIFY(DIFF P VAR) ¬=0 -> SIMPLIFY(DIFF Q VAR)¬=0 ->
574               FAILURE; POLYNOMIAL(%*,Q,P,)VAR;
575                  _NEW U,V,W,=POLYNOMIAL Q VAR
576                  _IN U -> U,V,MAKELIST V P,; FAILURE;
577         _HD S='**" ->
578          _NEW P,Q,=_TL S
579          _IN SIMPLIFY(DIFF P VAR)=0 -> _TRUE,0,(S,),;
580              P¬= VAR  -> FAILURE;
581              ¬_NUMBER Q -> FAILURE;
582              _TRUE, Q,MAKELIST Q 1,;
583    FAILURE
584    _AND ADD L M
585    _BE L=() -> ();
586        LENGTH L>LENGTH M -> SIMPLIFY(_HD L),ADD(_TL L) M;
587        SIMPLIFY(%+,_HD L,_HD M,),ADD(_TL L)(_TL M)
588    _AND MAKELIST DEGREE MEM
589    _BE DEGREE<0 -> ();
590        MEM,MAKELIST(DEGREE-1) 0
591    _AND NEGATE(X,Y,Z,)
592    _BE _LET NEGATE L
593        _BE L=() -> ();
594            _NUMBER(_HD L) -> -(_HD L),NEGATE(_TL L);
595            (%_,_HD L,),NEGATE (_TL L)
596        _IN X,Y,NEGATE Z,
597
598
599    _LET SQRT I  || THE SYMBOLIC SQUARE ROOT OF I
600    _BE STRING I -> 'SQRT",I,;
601        _NUMBER I -> _LET SQJ J
602                       _BE J*J=I -> J;
603                       J*J<I & (J+1)*(J+1)>I -> 'SQRT",I,;
604                       _NEW NEWJ=(J+I/J)/2 _IN SQJ NEWJ
605                       _IN SQJ 1;
606        _HD I=%* | _HD I=%/ -> SIMPLIFY(_HD I,SQRT(_HD _TL I),
607                                          SQRT(_HD _TL _TL I),);
608        _HD I='**" -> SIMPLIFY('**",_HD _TL I,(%/,_HD _TL _TL I,2,),);
609           'SQRT",I,
610
611    ||
612    ||
613    ||      INTEGRAL TABLES
614    ||
615    ||
616
```

```
_NEW EXPFORM (OP,LEFT,RIGHT,) VAR || INTEGRAL TABLES--EXPONENTIATIONS
_BE LEFT=( SEC ,VAR,) & RIGHT=2 ->  TAN ,VAR,;
    LEFT=( CSC ,VAR)&RIGHT=2  -> %_,( COT ,VAR,),;
    RIGHT=VAR -> NOTCONTAINS VAR LEFT ->
                 %/,(OP,LEFT,RIGHT,),('LN",LEFT,),;
    _FALSE;
     LEFT=VAR -> NOTCONTAINS VAR RIGHT ->
                 RIGHT=(%_,1,) -> 'LN",VAR,;
    %/,('**",VAR,(%+,1,RIGHT,),),(%+,1,RIGHT,),;
    _FALSE;
    _FALSE
_AND MULTFORM (OP,LEFT,RIGHT,) VAR || INTEGRAL TABLE--MULTIPLICATIONS
_BE LEFT=( TAN ,VAR,) & RIGHT=( SEC ,VAR,) ->  SEC ,VAR,;
    LEFT=( SEC ,VAR,) & RIGHT=( TAN ,VAR,) ->  SEC ,VAR,;
    LEFT=( CSC ,VAR,) & RIGHT=( COT ,VAR,) -> %_,( CSC ,VAR,),;
    LEFT=( COT ,VAR,) & RIGHT=( CSC ,VAR,) -> %_,( CSC ,VAR,),;
    _FALSE
_LET DIVFORM (OP,LEFT,RIGHT,) VAR || INTEGRAL TABLES--DIVISIONS
_BE NOTCONTAINS VAR (OP,LEFT,RIGHT,) -> %*,(OP,LEFT,RIGHT,),VAR,;
    LEFT¬=1 -> _FALSE; EQUAL RIGHT VAR -> 'LN",VAR,;
    _HD RIGHT=%+ | _HD RIGHT=%- ->
      _NEW A,B,C,=POLYNOMIAL RIGHT VAR
      _IN ¬A -> _FALSE;
          B¬=2 -> _FALSE;
          _LET P,Q,R,=C
          _IN Q¬=0 -> _FALSE;
              POS P & POS R ->
               %*,(%/,1,(%*,SQRT P,SQRT R,),),('ARCTAN",
                                (%/,(%*,SQRT P,VAR,),SQRT R,),),;
                POS P & ¬POS R ->
                _NEW R=_HD _TL R
              _IN  %*,(%/,1,(%*,2,(%*,SQRT P,SQRT R,),),),
                   ('LN",(%/,(%-,(%*,SQRT P,VAR,),SQRT R,),(%+,(%*,SQRT P,VAR,),
                                    SQRT R,),),),;
               ¬POS P & POS R ->
                _NEW P=_HD _TL P
                _IN %*,(%/,1,(%*,2,(%*,SQRT P,SQRT R,),),),
                     ('LN",(%/,(%+,(%*,SQRT P,VAR,),SQRT R,),
                                (%-,(%*,SQRT P,VAR,),SQRT R,),),),;
      _FALSE;
    _HD RIGHT='**" ->
     _HD _TL _TL RIGHT¬=(%/,1,2,) -> _FALSE;
      _LET A,B,C,=POLYNOMIAL(_HD _TL RIGHT) VAR
      _IN ¬A -> _FALSE;
```

```
661                 B=0 -> %*,(OP,LEFT,RIGHT,),VAR,;
662                 B=1 -> %*,2,RIGHT,;
663                 B¬=2 -> _FALSE;
664                 _NEW X,Y,Z,=C
665                 _IN Y¬=0 -> _FALSE;
666                     POS X -> %*,(%/,1,SQRT X,),('LN",(%+,(%*,SQRT X,VAR,),RIGHT,),),;
667                     POS Z -> _NEW X=_HD _TL X
668                              _IN %*,(%/,1,SQRT X,),('ARCSIN",(%/,(%*,SQRT X,VAR,),
669                                                               SQRT Z,),),;
670           _NEW A,B,C,=RIGHT
671           _IN B=VAR -> _NEW Z=SIMPLIFY(%-,C,1,)
672                        _IN %_,(%/,1,(%*,(A,B,Z,),Z,),),;
673           _FALSE;
674         _HD RIGHT=%* ->
675           _NEW P,Q,R,=RIGHT
676           _IN Q¬=VAR -> R¬=VAR -> _FALSE; DIVFORM(%/,1,(P,R,Q,),) VAR;
677               ¬_LIST R -> _FALSE;
678               _HD P¬='**" -> _FALSE;
679               _HD _TL _TL R ¬= (%/,1,2,)  -> _FALSE; || MUST BE SQRT FORM
680               _NEW X,Y,Z,=POLYNOMIAL(_HD _TL R) VAR
681               _IN ¬X -> _FALSE;
682                   Y¬=2 -> _FALSE;
683                   _NEW A,B,C,=Z
684                   _IN POS A & ¬POS C ->
685                        _NEW C=_HD _TL  C
686                       _IN %*,(%/,1,SQRT C,),('ARCSEC",(%/,(%*,SQRT A,VAR,),SQRT C,
687                            ),),;
688                 _FALSE;
689       _FALSE
690     _NEW SQFORM (OP,LEFT,RIGHT,) VAR || INTEGRAL TABLES--SQUARE ROOTS
691     _BE _NEW A,B,C,=POLYNOMIAL LEFT VAR
692         _IN ¬A -> _FALSE;
693             B=0 -> %*,(OP,LEFT,RIGHT,),VAR,;
694         B=1 -> %/,(%*,2,('**",LEFT,(%/,3,2,),),),3,;
695             B¬=2 -> _FALSE;
696             _NEW X,Y,Z,=C
697             _IN Y¬=0 -> _FALSE;
698                 POS X & ¬POS Z ->
699                 %*,(%/,1,SQRT X,),
700                    (%+,(%*,(%/,(%*,SQRT X,VAR,),2,),(OP,LEFT,RIGHT,),),
701                    (%*,(%/,Z,2,),('LN",  (%+,(%*,SQRT X,VAR,),(OP,LEFT,RIGHT,
702                          ),),),),),;
703                  POS X & ¬POS Z ->
704                   _NEW Z=_HD _TL Z
```

```
                    _IN  %*,(%/,1,SQRT X,),
                            (%-,(%*,(%/,(%*,SQRT X,VAR,),2,),(OP,LEFT,RIGHT,),),
                            (%*,(%/,Z,2,),('LN",(%+,(%*,SQRT X,VAR,),(OP,LEFT,RIGHT,
                              ),),),),),;
                    ¬POS X & POS Z ->
                      _NEW X=_HD _TL X
                      _IN %*,(%/,1,SQRT X,),
                              (%+,(%*,(%/,(%*,SQRT X,VAR,),2,),(OP,LEFT,RIGHT,),),
                              (%*,(%/,Z,2,),('ARCSIN",(%/,(%*,SQRT X,VAR,),SQRT Z,
                                ),),),),;
_FALSE
_NEW BASICFORM S VAR  ||    LOOK UP INTEGRAL TABLES
_BE _NUMBER S -> %*,S,VAR,;
    S=VAR -> %/,('**",VAR,2,),2,;
    STRING S -> %*,S,VAR,;
    _HD S='**" ->
                    _HD _TL _TL S=(%/,1,2,) -> SQFORM S VAR;
    EXPFORM S VAR;
    _HD S=%* -> MULTFORM S VAR;
    _HD S=%/ -> DIVFORM S VAR;
    S=( SIN ,VAR,) -> %_,( COS ,VAR,),;
    S=( COS ,VAR,) ->  SIN ,VAR,;
    S=( TAN ,VAR,) -> 'LN",( SEC ,VAR,),;
    S=( COT ,VAR,) ->  LN ,( SIN ,VAR,),;
    S=( SEC ,VAR,) -> 'LN",(%+,( SEC ,VAR,),( TAN ,VAR,),),;
    S=( CSC ,VAR,) -> 'LN",(%-,( CSC ,VAR,),( COT ,VAR,),),;
    S=('LN",VAR,) -> %-,(%*,VAR,('LN",VAR,),),VAR,;
    S=('LOG",VAR,) -> %*,('LOG",'E",),(%-,(%*,VAR,('LN",VAR,),),VAR,),;
    S=('EXPN",VAR,) -> S;
    _FALSE


||
||
||      METHODS OF INTEGRATION
||
||

_LET ARG FUNC VAR || FIND THE ARGUMENT OF AN OBJECT LESS MULT CONSTANTS
_BE NOTCONTAINS VAR FUNC -> ();
    _HD FUNC=%_ -> ARG(_HD _TL FUNC);
    _HD FUNC=%* -> SIMPLIFY(DIFF(_HD _TL FUNC)VAR)=0 ->
                      ARG(_HD _TL _TL FUNC)VAR;
    FUNC;
```

```
749         FUNC
750
751
752
753     _LET DIVIDE (NNUM,CONUM,) (NDEN,CODEN,) VAR || DIVISION OF POLYNOMIALS
754     _BE NNUM < NDEN -> (%/,ADDBACK CONUM VAR NNUM,ADDBACK CODEN VAR NDEN,);
755         _NEW QUOTIENT=SIMPLIFY(%/,_HD CONUM,_HD CODEN,)
756         _NEW PRODUCT=MULT QUOTIENT(APPEND(_TL CODEN)(ZEROES(NNUM-NDEN)))
757         _NEW REMAINDER=SUB(_TL CONUM) PRODUCT
758         _IN %+,SIMPLIFY(%*,QUOTIENT,('**",VAR,NNUM-NDEN,),),
759                                 DIVIDE(NNUM-1,REMAINDER,)(NDEN,CODEN,)VAR,
760     _AND ADDBACK (A,B) VAR N  || RECONSTRUCT POLYN FROM COEFFICENTS (A,B)
761     _BE N=0 -> A;
762         SIMPLIFY(%+,(%*,A,('**",VAR,N,),),ADDBACK B VAR (N-1),)
763     _AND MULT A L || MULTIPLY EACH MBR OF LIST L BY A
764     _BE L=() -> (); SIMPLIFY(%*,A,_HD L,),MULT A (_TL L)
765     _AND ZEROES N   || A LIST OF N ZEROES
766     _BE N=0 -> (); 0,ZEROES(N-1)
767     _AND SUB L M  || SUBTRACT NTH MBR OF LIST M FROM NTH OF L
768     _BE L=() -> ();
769         SIMPLIFY(%-,_HD L,_HD M,),SUB(_TL L)(_TL M)
770
771
772     _LET FACT(N)  || THE FACTORIAL OF N
773     _BE N=0 -> 1; N*FACT(N-1)
774     _NEW EXPAND(OP,(A,B,C,),P,)||  BINOMIAL EXPANSION
775     _BE _NEW X=B _AND Y= A=%- -> %_,C,; C
776         _LET EXPAND M
777         _BE M=0 -> '**",X,P,;
778             %+,(%*,FACT(P)/(FACT(M)*FACT(P-M)),
779                 (%*,('**",X,P-M,),('**",Y,M,),),), EXPAND(M-1),
780         _IN SIMPLIFY(EXPAND P)
781
782
783     _LET MULTIPLY X(OP,A,B,) ||MULTIPLY X INTO A SUM
784     _BE _NEW A= ATOMIC A -> %*,A,X,; _HD A=%+ | _HD A=%- -> MULTIPLY X A;
785                     %*,A,X,
786         _AND B= ATOMIC B -> %*,B,X,; _HD B=%+ | _HD B=%- -> MULTIPLY X B;
787                 %*,B,X,
788         _IN SIMPLIFY(OP,A,B,)
789
790
791     _AND FACCONST CONST S VAR  || FACTOR CONSTANTS FROM THE INTEGRAND
792     _BE _NEW S=SIMPLIFY(%/,S,CONST,)
    _IN _NEW A=INT S VAR
```

```
793                 _IN A=_FALSE -> _FALSE; %*,CONST,A,
794               _AND DIFFDIV A S VAR  || CHANGE-OF-VARIABLE SUBSTITUTION
795               _BE _NEW S=SIMPLIFY(%/,S,DIFF A VAR,)
796                 _IN INT S A
797               _AND INTPARTS S VAR ||INTEGRATION BY PARTS ROUTINE
798                 || FIRST CHECK FOR CASE EXPN(A*X)*SIN(B*X) OR EXPN(A*X)*COS(B*X)
799               _BE _HD S=%* -> _NEW (OP,LEFT,RIGHT,)=S
800                   _IN ¬ATOMIC LEFT & ¬ATOMIC RIGHT ->
801                         _HD LEFT='EXPN" & TRIGONOMETRIC RIGHT ->
802                          _HD RIGHT¬='SIN" | _HD RIGHT¬='COS" -> _FALSE;
803                          _NEW ARG1=_HD _TL LEFT _AND ARG2=_HD _TL RIGHT
804                          _NEW(A1,B1,C1,)=POLYNOMIAL ARG1 VAR
805                          _AND(A2,B2,C2,)=POLYNOMIAL ARG2 VAR
806                          _IN ¬(A1 & A2) -> _FALSE; ¬(B1=1 & B2=1) -> _FALSE;
807                              _NEW (A1,B1,)=C1 _AND (A2,B2,)=C2
808                               _IN ¬(B1=0 & B2=0) -> _FALSE;
809                                   _HD RIGHT='SIN" ->
810                                    %/,(%*,LEFT,(%-,(%*,A1,RIGHT,),(%*,A2,
811                                       ('COS",_HD _TL RIGHT,),),),),
812                                       (%-,(%*,A1,A1,),(%*,A2,A2,),),;
813                    || IF THE ABOVE FAILS, WE HAVE EXPN(AX)*COS(BX)
814                                    %/,(%*,LEFT,(%+,(%*,A1,RIGHT,),(%*,A2,
815                                       ('SIN",_HD _TL RIGHT,),),),),
816                                        (%*,(%*,A1,A1,),(%*,A2,A2,),),;
817                              TRY (PARTITIONS S VAR) VAR;
818                          TRY(PARTITIONS S VAR) VAR;
819                      TRY (PARTITIONS S VAR) VAR
820               _AND TRY L VAR
821               _BE L=() -> _FALSE;
822                   _NEW(U,DV,)=_HD L
823                   _NEW V=INT DV VAR
824                   _IN V=_FALSE -> TRY(_TL L)VAR;
825                   _NEW DU=SIMPLIFY(DIFF U VAR)
826                   _AND V=SIMPLIFY V
827                   _IN SAMEFORM (U,DV,)(V,DU,) VAR -> TRY (_TL L) VAR; || PARTITION FAILS
828                   _NEW A=INT(%*,V,DU,) VAR
829                   _IN A=_FALSE -> TRY(_TL L)VAR; %-,(%*,U,V,),A,
830               _AND SAMEFORM (U,DV,) (V,DU,) VAR  || IS AN 'INTPARTS' PARTITION USEFUL??
831               _BE ATOMIC U -> ¬ATOMIC DU;
832                   ATOMIC DV -> _TRUE;
833                   _TL _TL U=() ->
834                     _TL _TL DU¬=() -> _FALSE; _TRUE;
835                   _TL _TL DV=() ->
836                      _NEW I,J,K,=POLYNOMIAL U VAR
```

```
837                 _IN _TL _TL V=() -> I -> _FALSE; _TRUE; _FALSE;
838             _HD U='**' & _HD DV='**', -> _FALSE;
839         _TRUE
840         _AND PARTITIONS S VAR || FIND U AND DV FOR INTPARTS HEURISTICALLY
841         _BE _TL _TL S=() -> (S,1,),;
842             _NEW X=()
843             _NEW(OP,LEFT,RIGHT,)=S       _IN
844             OP=%* ->
845              _NEW X=ATOMIC LEFT & ¬ATOMIC RIGHT ->
846                       _TL _TL RIGHT=() -> PARTITIONS(OP,RIGHT,LEFT,)VAR;
847                     X; X
848              _NEW X=¬ATOMIC LEFT & ¬ATOMIC RIGHT ->
849                       (LEFT,RIGHT,),(RIGHT,LEFT,),X; X
850              _NEW X=_HD LEFT='LN" | _HD LEFT='ARCSIN" |
851                       _HD LEFT='ARCCOS" | _HD LEFT='ARCTAN" | _HD LEFT='ARCCOT" |
852                       _HD LEFT='ARCSEC" | _HD LEFT='ARCCSC" -> (LEFT,RIGHT,),X; X
853              _NEW X= TRIGONOMETRIC LEFT ->
854                       (SIMPLIFY(%/,RIGHT,DIFF(_HD _TL LEFT)VAR,),LEFT,),X; X
855              _NEW X=_HD LEFT='EXPN" ->
856                         _NEW I,J,K,=POLYNOMIAL RIGHT VAR
857                         _IN ¬I -> TRIGONOMETRIC RIGHT -> X;
858                              (SIMPLIFY(%/,RIGHT,DIFF(_HD _TL LEFT)VAR,),LEFT,),X; X; X
859              _NEW X= _HD LEFT='**" & _HD RIGHT='**" ->
860                        TRIGONOMETRIC(_HD _TL LEFT) ->
861                          (SIMPLIFY(%/,RIGHT,DIFF(_HD _TL _HD _TL LEFT) VAR,),LEFT,),X;
862                       TRIGONOMETRIC(_HD _TL RIGHT) ->
863                       (SIMPLIFY(%/,LEFT,DIFF(_HD _TL _HD _TL RIGHT)VAR,),RIGHT,),X;
864                       ¬ATOMIC(_HD _TL _TL LEFT) ->
865                          (SIMPLIFY(%/,RIGHT,DIFF(_HD _TL LEFT)VAR,),LEFT,),X;
866                       ¬ATOMIC(_HD _TL _TL RIGHT) ->
867                          (SIMPLIFY(%/,LEFT,DIFF(_HD _TL RIGHT)VAR,),RIGHT,),X;
868                       ¬_NUMBER(_HD _TL _TL RIGHT) & ¬_NUMBER(_HD _TL _TL LEFT) -> X;
869                       (_HD _TL _TL RIGHT)-(_HD _TL _TL LEFT)<0 ->
870                        (SIMPLIFY(%/,RIGHT,DIFF(_HD _TL LEFT)VAR,),LEFT,),X;
871                        (SIMPLIFY(%/,LEFT,DIFF(_HD _TL RIGHT)VAR,),RIGHT,),X;
872               X
873              _NEW X=_HD RIGHT='**"
874                     -> ¬_NUMBER(_HD _TL _TL RIGHT) -> X;
875                        ATOMIC LEFT -> (LEFT,RIGHT,),X;
876                        (RIGHT,LEFT,),X;
877                     X
878             _IN
879               _HD LEFT='**" & _HD RIGHT¬='**" -> PARTITIONS(OP,RIGHT,LEFT,)VAR;
880              REVERSE X;
```

```
881         OP='/ ->
882          _NEW X=_HD RIGHT='**" ->
883            _NEW A=DIFF(_HD _TL RIGHT)VAR
884            _IN (SIMPLIFY('/,LEFT,A,),('/,A,RIGHT,),),X;
885            _HD RIGHT='**" & _HD LEFT='**" ->
886              (RIGHT,('/,1,LEFT,),),X;
887          X
888         _IN REVERSE X;
889          OP='**" ->
890           STRING RIGHT -> (); (S,1,),(LEFT,(OP,LEFT,SIMPLIFY('-,RIGHT,1,),),),;
891          (S,1,),
892    _AND ARCSUB S VAR || SPECIAL SUBSTITUTION FOR RATIONAL SIN AND COS FUNC.
893    _BE _NEW DENOM='+,1,('**" ,'Z",2,),
894         _NEW S=SUBST('/,('*,2,'Z",),DENOM,)( SIN ,VAR,) S
895         _NEW S=SUBST('/,('-,1,('**",'Z",2,),),DENOM,)( COS ,VAR,) S
896         _NEW S=SUBST('*,2,('ARCTAN",'Z",),) VAR S
897         _NEW A=INT('*,S,('/,2,DENOM,),) 'Z"
898         _IN A=_FALSE -> _FALSE;
899             SUBST   ( TAN ,('/,VAR,2,),) 'Z" A
900    _AND COMPSQUARE (A,B,C,) EXP S VAR ||  COMPLETING THE SQUARE
901    _BE B=0 -> _FALSE;
902       ||     (A,B,C,)=COEFF. OF EXP TO BE COMPLETED, EXP=FULL EXP TO COMPLETE
903         _NEW NEWC=SIMPLIFY('/,('*,B,B,),('*,4,A,),)
904         _NEW NEWC1=SIMPLIFY('-,C,NEWC,)
905         _NEW NEWCSQ=POS NEWC -> SQRT NEWC; '_,SQRT(_HD _TL NEWC),
906         _NEW NEWEXP=POS A ->
907             POS B ->
908               SIMPLIFY('+,('*,SQRT A,VAR,),NEWCSQ,);
909               SIMPLIFY('-,('*,SQRT A,VAR,),NEWCSQ,);
910             POS B ->
911              SIMPLIFY('+,('*,SQRT(_HD _TL A),VAR,),NEWCSQ,);
912            .SIMPLIFY('-,('*,SQRT(_HD _TL A),VAR,),NEWCSQ,)
913         _IN POS A ->
914             BASICFORM(SUBST('+,('**",NEWEXP,2,),NEWC1,) EXP S) NEWEXP;
915             BASICFORM(SUBST('-,NEWC1,('**",NEWEXP,2,),) EXP S) NEWEXP
916    _AND EXPANSION EXP S VAR  ||  EXPAND BINOMIALS FOR INTEGRATION
917    _BE _NEW S=SUBST(EXPAND EXP) EXP S
918         _IN INT S VAR
919    _AND SEPARATE (OP,LEFT,RIGHT,) VAR  ||  INTEGRATE TERMS SEPARATELY (OP=/)
920    _BE _LET SEPARATE S
921         _BE ATOMIC S -> INT (OP,S,RIGHT,) VAR;
922             ¬(_HD S='+ | _HD S='-) -> INT(OP,S,RIGHT,) VAR;
923             _NEW(A,X,Y,)=S
924             _NEW I=SEPARATE X _AND J=SEPARATE Y
```

```
925             _IN I=_FALSE | J=_FALSE -> _FALSE;
926                   A,I,J,
927         _IN SEPARATE LEFT
928     _AND MULTEXPAND A (OP,LEFT,RIGHT,) S VAR  || MULT A POLYNOMIAL THRU BY A
929     _BE ¬_NUMBER RIGHT -> _FALSE;
930         INT  (MULTIPLY A (EXPAND (OP,LEFT,RIGHT,) ) ) VAR
931
932
933     _AND NEST (OP,L,R,) S VAR  || HANDLE (OP,L,R,) AS ARG OF FUNCTION
934     _BE _NEW PVAR='X¬"
935         _NEW INVERT=SIMPLIFY("/,1,R,)
936         _IN _NUMBER R -> _NEW X=SUBST PVAR (OP,L,R,) S
937                          _NEW X=SIMPLIFY("/,X,DIFF(OP,L,R,)VAR,)
938                          _NEW X=SUBST('**",PVAR,INVERT,) VAR X
939                          _NEW ANS=INT X PVAR
940                          _IN ANS=_FALSE -> ANS;
941                             SUBST ('**",VAR,INVERT,) PVAR ANS;
942         ATOMIC R -> _FALSE;
943         ¬(_HD R="/) -> _FALSE;
944         _NEW X=SUBST('**",PVAR,INVERT,) VAR S
945         _NEW X="*,DIFF('**",PVAR,INVERT,)PVAR,X,
946         _NEW ANS=INT X PVAR
947         _IN ANS=_FALSE -> _FALSE;
948             SUBST(OP,L,R,) PVAR ANS
949     _AND RADSUB  (A,B,)  S  VAR   ||  HANDLE RADICAL OF VAR IN INTEGRAND
950     _BE _NEW PVAR='X¬"
951         _NEW INVERT=SIMPLIFY("/,1,B,)
952         _NEW DU="*,INVERT,("*,DIFF A VAR,PVAR,),
953         _NEW P,Q,(R,T,),=POLYNOMIAL A VAR
954         _NEW X=SUBST('**",PVAR,INVERT,)  A  S
955         _NEW X=SUBST("/,("-,('**",PVAR,INVERT,),T,),R,) VAR X
956         _NEW ANS=INT ("*,X,DU,) PVAR
957         _IN ANS=_FALSE -> _FALSE;
958             SUBST('**",A,B,) PVAR ANS
959     _AND POLYDIV A B S VAR || LONG DIVISION OF POLYNOMIALS
960     _BE _LET X=DIVIDE A B VAR
961         _LET TERMWISE L
962         _BE ATOMIC L -> INT L VAR;
963             ¬(_HD L="+ | _HD L="-) -> INT L VAR;
964             _NEW OP,LEFT,RIGHT,=L
965             _NEW M=TERMWISE LEFT _AND N=TERMWISE RIGHT
966             _IN M= _FALSE -> _FALSE;
967                 N=_FALSE -> _FALSE; OP,M,N,
968             _IN TERMWISE X
```

```
969
970             _AND SPECIALFORM2 S VAR  ||  POWER OR ROOT OF VAR NESTED IN A FUNC?
971             _BE ATOMIC S -> _FALSE,(),;
972                 _HD S=/* -> _LET H,T,=_TL S
973                             _LET A,X,=SPECIALFORM2 H VAR
974                             _AND B,Y,=SPECIALFORM2 T VAR
975                             _IN A -> A,X,;
976                                 B,Y,;
977                 _TL _TL S ¬= () -> _FALSE,(),;
978                 _LET X=_HD _TL S
979                 _IN _HD X ¬= '**" -> _FALSE,(),;
980                     _HD _TL X ¬= VAR -> _FALSE,(),;
981                     _TRUE,X,
982
983             _AND SPECIALFORM3 S VAR || IS A SUBST FOR RADICAL APPROPRIATE?
984             _BE S=() -> _FALSE,(),(),;
985                ATOMIC S -> _FALSE,(),(),;
986                 _HD S='**" -> _LET OP,R,L,=S
987                               _IN ATOMIC L -> _FALSE,(),(),;
988                                   _HD L ¬=*/ -> _FALSE,(),(),;
989                                   _LET A,D,C,=POLYNOMIAL R VAR
990                                   _IN ¬A -> _FALSE,(),(),;
991                                       D¬=1 -> _FALSE,(),(),;
992                 _TRUE,R,L,; || RETURN EXPRESSION UNDER RADICAL ALONG WITH ROOT
993                 _TL _TL  S = ()  -> _FALSE,(),(),;
994                 _LET A,X,M,=SPECIALFORM3 (_HD _TL _TL S) VAR
995                 _AND B,Y,N,=SPECIALFORM3 (_HD _TL S) VAR
996                 _IN A -> A,X,M,;
997                     B,Y,N,
998
999             _AND SPECIALFORM S VAR   || IS A TRIG SUBSTITUTION APPROPRIATE FOR S?
1000            _BE S=() -> FAILURE; ATOMIC S -> FAILURE; _TL _TL S=() -> FAILURE;
1001                _HD S='**" ->
1002                 _LET P,DEGREE,COEF,=POLYNOMIAL(_HD _TL S) VAR
1003                 _IN ¬P | DEGREE¬=2 -> FAILURE;
1004                 _TRUE,S,COEF,;  ||  RETURN EXPRESSION TO BE SUBSTITUTED FOR WITH
1005                                 ||  ITS POLYNOMIAL COEFFICIENTS
1006                 _NEW A,X,M,=SPECIALFORM(_HD _TL S) VAR
1007                 _AND B,Y,N,=SPECIALFORM(_HD _TL _TL S) VAR
1008                 _IN A -> A,X,M,;
1009                     B -> B,Y,N,;
1010                FAILURE
1011            _AND TRIGSUB(OP,LEFT,RIGHT,)(A,B,C,) S VAR          ||TRIG SUBSTITUTIONS
1012            _BE B¬=0 -> COMSQ(OP,LEFT,RIGHT,)(A,B,C,) S VAR;
```

```
_LET PVAR='X¬"
_LET A1 = POS A -> SQRT A; SQRT(_HD _TL A)
_LET C1 = POS C -> SQRT C; SQRT(_HD _TL C)
_LET CONSTANTS  ||  THE CONSTANTS PRODUCED BY THE BELOW SUBSTITUTIONS
_BE _NEW Z=SUBST('**",C1,(%*,2,RIGHT,),) (OP,LEFT,RIGHT,) S
    _NEW Z=SUBST(%/,C1,A1,) VAR Z
    _IN %*,(%/,C1,A1,),Z,
_LET OPER  || HELP FOR SIMPLIFYING THE D(PVAR) INTO THE SUBSTITUTION
_BE _NEW SIGN,L,R,=S
    _LET FIND OBJECT  ||  FIND OUT IF (OP,LEFT,RIGHT,) IS ABOVE OR
                      ||  BELOW A DIVISION SIGN
    _BE OBJECT=() -> _FALSE;
        ATOMIC OBJECT -> _FALSE;
        OBJECT=(OP,LEFT,RIGHT,) -> _TRUE;
        FIND(_HD OBJECT) | FIND (_TL OBJECT)
    _IN SIGN¬=%/ -> %*;
        FIND L -> %*;
        FIND R -> %/;
        'CHECK4"
_IN POS A & POS C
    -> _NEW S=SUBST(OPER,('**",(SEC ,PVAR,),2,),
                         ('**",(SEC,PVAR,),(%*,2,RIGHT,),),)
                    (OP,LEFT,RIGHT,) S
       _NEW S=SUBST(TAN,PVAR,) VAR S
       _NEW I=INT S PVAR
       _IN I=_FALSE -> _FALSE;
           _NEW OPP=(%*,A1,VAR,) _AND ADJ=C1
           _AND HYP= OP,LEFT,(%/,1,2,),
           _NEW S=RESUB OPP ADJ HYP I PVAR
           _NEW S=SUBST('ARCTAN",VAR,) PVAR S
           _IN %*,CONSTANTS,S,;
    ¬POS A & POS C
    -> _NEW S= SUBST(OPER,(COS,PVAR,),
                         ('**",(COS,PVAR,),(%*,2,RIGHT,),),)
                    (OP,LEFT,RIGHT,)  S
       _NEW S=SUBST(SIN,PVAR,) VAR S
       _NEW I=INT S PVAR
       _IN I=_FALSE -> _FALSE;
           _NEW OPP=(%*,A1,VAR,) _AND HYP=C1
           _AND ADJ= OP,LEFT,(%/,1,2,),
           _NEW S=RESUB OPP ADJ HYP I PVAR
           _NEW S=SUBST('ARCSIN",VAR,) PVAR S
           _IN %*,CONSTANTS,S,;
    POS A & ¬POS C
```

```
            -> _NEW HOLD=SUBST 1 (OP,LEFT,RIGHT,) S
               _NEW HOLD=SIMPLIFY(SUBST (SEC,PVAR,) VAR HOLD)
               _NEW HOLD=%*,(SEC,PVAR,),HOLD, || MULTIPLY PART OF D(PVAR)
               _NEW HOLD1=OPER,(TAN,PVAR,),('**",(TAN,PVAR,),(%*,2,RIGHT,),),
               _NEW I=INT(%*,HOLD,HOLD1,)PVAR
               _IN I=_FALSE -> _FALSE;
                    _NEW HYP=%*,A1,VAR, _AND ADJ= C1
                    _AND OPP=OP,LEFT,(%/,1,2,),
                    _NEW S=RESUB OPP ADJ HYP I PVAR
                    _NEW S=SUBST('ARCSEC",VAR,) PVAR S
                    _IN %*,CONSTANTS,S,;
    _FALSE
_AND COMSQ (OP,LEFT,RIGHT,) (A,B,C,) S VAR
_BE _LET NEWC=SIMPLIFY(%/,(%*,B,B,),(%*,4,A,),)
    _LET NEWCSQ=POS NEWC -> SQRT NEWC; %_,SQRT(_HD _TL NEWC),
    _LET NEWEXP=
       POS A ->
        POS B -> SIMPLIFY(%+,(%*,SQRT A, VAR,),NEWCSQ,);
                 SIMPLIFY(%-,(%*,SQRT A,VAR,),NEWCSQ,);
        POS B -> SIMPLIFY(%+,(%*,SQRT(_HD _TL A),VAR,),NEWCSQ,);
       SIMPLIFY(%-,(%*,SQRT(_HD _TL A),VAR,),NEWCSQ,)
    _LET NEWLEFT=
        POS A -> SIMPLIFY(%+,(%-,C,NEWC,),('**",NEWEXP,2,),);
        SIMPLIFY(%-,(%-,C,NEWC,),('**",NEWEXP,2,),)
    _LET T,D,CO,=POLYNOMIAL NEWLEFT NEWEXP
    _NEW S=SUBST NEWLEFT LEFT S
    _IN TRIGSUB NEWLEFT   CO  S  NEWEXP
_AND RESUB OPP ADJ HYP S VAR
_BE _NEW S=SUBST(%/,OPP,HYP,)( SIN ,VAR,) S
    _NEW S=SUBST(%/,ADJ,HYP,)( COS ,VAR,) S
    _NEW S=SUBST(%/,OPP,ADJ,)( TAN ,VAR,) S
    _NEW S=SUBST(%/,ADJ,OPP,)( COT ,VAR,) S
    _NEW S=SUBST(%/,HYP,ADJ,)( SEC ,VAR,) S
    _NEW S=SUBST(%/,HYP,OPP,)( CSC ,VAR,) S
    _NEW S=SUBST(%*,2,(%*,(%/,ADJ,HYP,),(%/,OPP,HYP,),),)
             ( SIN ,(%*,2,VAR,),) S
    _NEW S= SUBST(%-,('**",(%/,ADJ,HYP,),2,),('**",(%/,OPP,HYP,),2,),)
              ( COS ,(%*,2,VAR,),) S
_IN S

 _AND TRIGONOMETRIC S  ||  IS S TRIGONOMETRIC???
_BE ATOMIC S -> _FALSE; S=() -> _FALSE;
    _HD S= SIN  | _HD S= COS  | _HD S= TAN  |
    _HD S= COT  | _HD S= SEC  | _HD S= CSC   -> _TRUE;
```

```
      TRIGONOMETRIC(_HD S) | TRIGONOMETRIC(_TL S)
_AND INTRIG S VAR || INTEGRATION OF EXPRESSIONS CONTAINING TRIG FUNCTIONS
_BE _HD S="**" -> TRIGEXP S VAR; _HD S=%/ -> TRIGDIV S VAR;
    _HD S=%* -> TRIGMULT S VAR;
    _FALSE
_AND TRIGEXP  (OP,X,Y,) VAR
_BE _HD X= SIN  | _HD X= COS  | _HD X=  TAN  |
    _HD X= COT  | _HD X= SEC  | _HD X= CSC  ->
    _NEW A,B,=X
    _IN ¬POS Y ->
         _NEW S=OP,X,_HD _TL Y,
         _IN _NEW S=
              A= SIN  -> SUBST  CSC  X S;
              A= COS  -> SUBST  SEC  X S;
              A= TAN  -> SUBST  COT  X S;
              A= COT  -> SUBST  TAN  X S;
              A= SEC  -> SUBST  COS  X S;
              A= CSC  -> SUBST  SIN  X S;  CHECK1
            _IN TRIGEXP S VAR;
   ¬_NUMBER Y -> _FALSE;
   Y=2 ->
    A= SEC  | A= CSC  -> _FALSE; ||SHOULD HAVE BEEN HANDLED ELSEWHERE
    A=SIN -> INT (%/,(%-,1,(COS,(%*,2,B,),),),2,)  VAR;
    A=COS -> INT (%/,(%+,1,(COS,(%*,2,B,),),),2,)  VAR;
    A=TAN -> INT (%-,1,("**",(SEC,B,),2,),)  VAR;
    A=COT -> INT (%-,1,("**",(CSC,B,),2,),)  VAR;
   _FALSE;
   _NEW P=INT(OP,X,Y-2,) VAR
   _IN P=_FALSE -> _FALSE;
       A= SIN  ->
        %+,(%_,(%/,(%*,("**",(A,B,),Y-1,),( COS ,B,),),Y,),),
                 (%*,(%/,Y-1,Y,),P,),;
       A= COS  ->
        %+,(%/,(%*,("**",(A,B,),Y-1,),( SIN ,B,),),Y,),
           (%*,(%/,Y-1,Y,),P,),;
       A= TAN  ->
         %-,(%/,("**",(A,B,),Y-1,),Y-1,),P,;
       A= COT  ->
         %-,(%_,(%/,("**",(A,B,),Y-1,),Y-1,),),P,;
       A= SEC  ->
         %+,(%/,(%*,("**",(A,B,),Y-2,),("TAN",B,),),Y-1,),
            (%*,(%/,Y-2,Y-1,),P,),;
       A= CSC  ->
        %+,(%-,(%/,(%*,("**",(A,B,),Y-2,),( COT ,B,),),Y-1,),),
```

```
1145                (%*,(%/,Y-2,Y-1,),P,),;
1146        _FALSE; _FALSE
1147    _AND FINDARG S  ||  FIND THE ARGMENT OF ANY TRIG EXPRESSION IN S
1148    _BE _TL _TL S=() -> _HD _TL S;
1149        _NEW A,X,Y,=S
1150        _IN ¬_LIST X -> FINDARG Y;
1151            X=() -> FINDARG Y;
1152            _HD X= SIN  | _HD X= COS  | _HD X= TAN  |
1153            _HD X= COT  | _HD X= SEC  | _HD X= CSC  -> _HD _TL X;
1154        FINDARG Y
1155    _AND TRIGDIV (OP,LEFT,RIGHT,) VAR
1156    _BE ATOMIC LEFT & _TL _TL RIGHT=() ->
1157         _NEW FUNC=
1158          _HD RIGHT= SIN  ->  CSC ,_TL RIGHT;
1159          _HD RIGHT= COS  ->  SEC ,_TL RIGHT;
1160          _HD RIGHT= TAN  ->  COT ,_TL RIGHT;
1161          _HD RIGHT= COT  ->  TAN ,_TL RIGHT;
1162          _HD RIGHT= SEC  ->  COS ,_TL RIGHT;
1163          _HD RIGHT= CSC  ->  SIN ,_TL RIGHT; CHECK2
1164         _IN INT(%*,LEFT,FUNC,) VAR;
1165        _NUMBER LEFT & _HD RIGHT='**" ->
1166         ¬TRIGONOMETRIC(_HD _TL RIGHT) | ¬ _NUMBER(_HD _TL _TL RIGHT) ->
1167           _NEW (A,X,S,)=RIGHT
1168           _IN _TL _TL X¬=() -> _FALSE;
1169               _NEW FUNC=
1170                _HD X= SIN  ->  CSC ,_TL X;
1171                _HD X= COS  ->  SEC ,_TL X;
1172                _HD X= TAN  ->  COT ,_TL X;
1173                _HD X= COT  ->  TAN ,_TL X;
1174                _HD X= SEC  ->  COS ,_TL X;
1175                _HD X= CSC  ->  SIN ,_TL X; CHECK3
1176               _IN INT(%*,LEFT,('**",FUNC,S,),) VAR;
1177         _FALSE;
1178        _NEW PATT=PATTERN(OP,LEFT,RIGHT,)
1179        _IN _FUNCTION PATT -> PATT(OP,LEFT,RIGHT,) VAR;
1180        _NEW A=FINDARG(OP,LEFT,RIGHT,)
1181        _NEW S=CONVERT( SIN ,A,) (OP,LEFT,RIGHT,)
1182        _IN ARCSUB S VAR
1183    _AND TRIGMULT (OP,LEFT,RIGHT,) VAR  || MULT CONTAINING TRIG EXPRESSION
1184    _BE _NEW P=PATTERN(OP,LEFT,RIGHT,)
1185        _IN ¬_NUMBER P -> P(OP,LEFT,RIGHT,) VAR;
1186            ATOMIC LEFT | ATOMIC RIGHT -> _FALSE;
1187            ¬TRIGONOMETRIC LEFT | ¬TRIGONOMETRIC RIGHT ->
1188              INTPARTS(OP,LEFT,RIGHT,) VAR;
```

```
1189                _NEW ARGUMENT=
1190                  _HD LEFT='**" & TRIGONOMETRIC (_HD _TL LEFT) ->
1191                       _HD _TL _HD _TL LEFT; _HD _TL LEFT
1192                  _NEW S=CONVERT( SIN ,ARGUMENT,) (OP,LEFT,RIGHT,)
1193                  _IN EQUAL S (OP,LEFT,RIGHT,) -> _FALSE; ||WE DIDN'T GET ANYWHERE
1194                      INT S VAR
1195
1196
1197          _AND CONVERT(FUNC,ARG,) S ||  THE TRIG IDENTITIES
1198          _BE FUNC= SIN  ->
1199               _NEW S=SUBST(%/,( SIN ,ARG,),( COS ,ARG,),)( TAN ,ARG,) S
1200               _NEW S=SUBST(%/,( COS ,ARG,),( SIN ,ARG,),)( COT ,ARG,) S
1201               _NEW S=SUBST(%/,1,( COS ,ARG,),) ( SEC ,ARG,) S
1202               _NEW S=SUBST(%/,1,( SIN ,ARG,),) ( CSC ,ARG,)   S
1203               _IN S;
1204              FUNC= TAN  ->
1205               _NEW S=SUBST          ('**",(%+,('**",( TAN ,ARG,),2,),1,),(%/,1,2,),)
1206                          ( SEC ,ARG,)     S
1207               _NEW S=SUBST(%/,1,( TAN ,ARG,),), ( COT ,ARG,) S
1208               _IN S;
1209              FUNC= COT  ->
1210                 _NEW S=SUBST ('**",(%+,('**",( COT ,ARG,),2,),1,),(%/,1,2,),)
1211                            ( CSC ,ARG,) S
1212                 _NEW S=SUBST(%/,1,( COT ,ARG,),) ( TAN ,ARG,) S
1213                 _IN S;
1214              FUNC= SEC  ->
1215                _NEW S=SUBST('**",(%-,('**",( SEC ,ARG,),2,),1,),(%/,1,2,),)
1216                                 ( TAN ,ARG,) S
1217                _IN SUBST(%/,1,( SEC ,ARG,),) ( COS ,ARG,) S;
1218              FUNC= CSC  ->
1219                _NEW S=SUBST('**",(%-,('**",( CSC ,ARG,),2,),1,),(%/,1,2,),)
1220                                 ( COT ,ARG,) S
1221                _IN SUBST(%/,1,( CSC ,ARG,),) ( SIN ,ARG,) S;
1222          S
1223
1224
1225          _AND PATTERN S  ||  RECOMMEND COURSE OF ACTION FOR TRIG EXPRESSION
1226          _BE _TL _TL S=() -> 0;
1227               _NEW OP,LEFT,RIGHT,=S _IN
1228               OP=%/ -> PATTERN(%*,LEFT,SIMPLIFY('**",RIGHT,(%_,1,),),);
1229               OP=%* ->
1230                _NEW(OP,LEFT,RIGHT,)=S
1231                _IN _HD LEFT= SIN  ->
1232                      _HD RIGHT= SIN  -> SINSIN;
```

```
1233                _HD RIGHT= COS  ->
1234                    _HD _TL LEFT=_HD _TL RIGHT ->    0    ; SINCOS2;
1235          0;
1236          _HD RIGHT= SIN  -> PATTERN(OP,RIGHT,LEFT,);
1237          _NEW X=_HD LEFT ='**" -> _HD _TL LEFT; LEFT
1238          _AND Y=_HD RIGHT='**" -> _HD _TL RIGHT; RIGHT
1239          _IN _HD X= SIN  ->
1240               Y=( COS ,_HD _TL X,) -> REDSIN;
1241            0;
1242          _HD Y= SIN  -> PATTERN(OP,RIGHT,LEFT,);
1243          _HD X= TAN  ->
1244           Y=( SEC ,_TL X) -> REDTAN;
1245          0;
1246          _HD Y= TAN  -> PATTERN(OP,RIGHT,LEFT,);
1247          _HD X= COT  ->
1248            Y=( CSC ,_TL X) -> REDCOT;
1249           0;
1250          _HD Y= COT  -> PATTERN(OP,RIGHT,LEFT,);
1251          0;
1252        0
1253
1254
1255  || REDUCTION ROUTINES FOR TRIGONOMETRIC EXPRESSIONS
1256  _AND SINSIN(OP,(A,B,),(X,Y,),) VAR
1257  _BE INT(%*,(%/,1,2,),(%-,( COS ,(%-,B,Y,),),( COS ,(%+,B,Y,),),),) VAR
1258  _AND SINCOS2 (OP,(A,B,),X,Y,)    VAR
1259  _BE INT(%*,(%/,1,2,),(%+,( SIN ,(%-,B,Y,),),( SIN ,(%+,B,Y,),),),) VAR
1260  _AND REDSIN (OP,LEFT,RIGHT,) VAR
1261  _BE _NEW A=_HD LEFT='**" -> _HD _TL LEFT; LEFT
1262       _AND X=_HD RIGHT='**" -> _HD _TL RIGHT; RIGHT
1263       _IN _HD A= COS  -> REDSIN (OP,RIGHT,LEFT,) VAR;
1264            _TL A¬=_TL X -> _FALSE;
1265            _HD _TL A ¬= VAR -> _NEW P,Q,R,=POLYNOMIAL(_HD _TL A) VAR
1266                                _IN ¬P -> _FALSE;
1267                                    Q¬=1 -> _FALSE;
1268                                    DIFFDIV(_HD _TL A)(OP,LEFT,RIGHT,)VAR;
1269            _NEW M=_HD LEFT= SIN  -> 1; _HD _TL _TL LEFT
1270            _AND N=_HD RIGHT= COS  -> 1; _HD _TL _TL RIGHT
1271            _IN SIMPLIFY(%+,M,N,)=0 ->
1272                  ¬POS M -> INT ('**",(COT,_TL X),N,) VAR;
1273                  INT ('**",(TAN,_TL A),M,) VAR;
1274               (¬_NUMBER M | ¬POS M) & (¬_NUMBER N | ¬POS N) -> _FALSE;
1275               ¬_NUMBER M & ¬_NUMBER N  -> _FALSE;
1276               ¬_NUMBER N | ¬POS N ->
```

```
1277                    M _MOD 2=1 ->
1278                     INT(MULTIPLY RIGHT(EXPAND
1279                      ('**",(%- ,1,('**",(%-,1,X,),2,),),(M-1)/2,)))  X;
1280                    INT('/,EXPAND('**",(%-,1,('**",      X      ,2,),),M/2,),
1281                      ('**",      X      ,(%_,N,),),) VAR;
1282                ¬_NUMBER M | ¬POS M ->
1283                 N _MOD 2=1 ->
1284                  INT(%_,(MULTIPLY LEFT(EXPAND
1285                    ('**",(%-,1,('**",      A      ,2,),),(N-1)/2,))),) A;
1286                  INT(%/,EXPAND('**",(%-,1,('**",      A      ,2,),),N/2,),
1287                    ('**",      A      ,(%_,N,),),) VAR;
1288                M _MOD 2=1 ->
1289                 INT(MULTIPLY RIGHT (EXPAND
1290                  ('**",(%-,1,('**",      X      ,2,),),(M-1)/2,))) X;
1291                N _MOD 2=1 ->
1292                 INT(%_,(MULTIPLY LEFT(EXPAND
1293                    ('**",(%-,1,('**",      A      ,2,),),(N-1)/2,))),) A;
1294            INT(MULTIPLY LEFT (EXPAND('**",(%-,1,('**",      A      ,2,),),),
1295                  N/2,)))VAR
1296    _AND REDTAN (OP,LEFT,RIGHT,)VAR
1297    _BE _NEW A=_HD LEFT='**" -> _HD _TL LEFT; LEFT
1298        _AND X=_HD RIGHT='**" -> _HD _TL RIGHT; RIGHT
1299        _IN _HD A= SEC  -> REDTAN (OP,RIGHT,LEFT,) VAR;
1300            _TL A¬= _TL X -> _FALSE;
1301            _HD _TL A ¬= VAR -> _NEW P,Q,R,=POLYNOMIAL(_HD _TL A) VAR
1302                                _IN ¬P -> _FALSE;
1303                                   Q¬=1 -> _FALSE;
1304            DIFFDIV(_HD _TL A) (OP,LEFT,RIGHT,) VAR;
1305            _NEW M=_HD LEFT= TAN  -> 1; _HD _TL _TL LEFT
1306            _AND N=_HD RIGHT= SEC  -> 1; _HD _TL _TL RIGHT
1307                _IN (¬POS M | ¬_NUMBER M) & (¬POS N | ¬_NUMBER N) -> _FALSE;
1308                    ¬_NUMBER M | ¬POS M  ->
1309                      N _MOD 2 =0 ->
1310                        INT(MULTIPLY LEFT
1311                          (EXPAND('**",(%+,('**",A,2,),1,),(N-2)/2,))) A;
1312                        INT(CONVERT('SIN",_TL A) (OP,LEFT,RIGHT,)) VAR;
1313                 ¬_NUMBER N | ¬POS N  ->
1314                    INT( CONVERT ( SIN ,_TL A)  (OP,LEFT,RIGHT,) ) VAR;
1315                 N _MOD 2=0 -> INT (MULTIPLY LEFT
1316                                  (EXPAND('**",(%+,1,('**",A,2,),),(N-2)/2,
1317                                   ))) A;
1318                 M _MOD 2=1 -> INT(MULTIPLY ('**",X,N-1,)
1319                                  (EXPAND('**",(%-,('**",X,2,),1,),(M-1)/2,)))
1320                                   X;
```

```
                    INT(MULTIPLY RIGHT
                        (EXPAND('**",(%-,('**",X,2,),1,),M/2,))) VAR
_AND REDCOT (OP,LEFT,RIGHT,) VAR
_BE _NEW A=_HD LEFT='**" -> _HD _TL LEFT; LEFT
    _AND X=_HD RIGHT='**" -> _HD _TL RIGHT; RIGHT
    _IN _HD A= CSC  -> REDCOT (OP,RIGHT,LEFT,) VAR; ||CANONICAL ORDER
        _TL A¬= _TL X -> _FALSE;
        _HD _TL A ¬= VAR -> _NEW P,Q,R,=POLYNOMIAL (_HD _TL A) VAR
                            _IN ¬P -> _FALSE;
                                Q¬=1 -> _FALSE;
                                DIFFDIV(_HD _TL A) (OP,LEFT,RIGHT,) VAR;
        _NEW M=_HD LEFT= COT  -> 1; _HD _TL _TL LEFT
        _AND N=_HD RIGHT= CSC  -> 1; _HD _TL _TL RIGHT
        _IN (¬POS M | ¬_NUMBER M) & (¬POS N | ¬_NUMBER N) -> _FALSE;
            ¬_NUMBER M | ¬POS M ->
              N _MOD 2 =0 ->
                INT(MULTIPLY LEFT
                    (EXPAND('**",(%+,('**",A,2,),1,),(N-2)/2,))) A;
            INT(CONVERT ('SIN",_TL A)(OP,LEFT,RIGHT,)) VAR;
            ¬_NUMBER N | ¬POS N
               -> INT(CONVERT( SIN ,_TL A) (OP,LEFT,RIGHT,)) VAR;
            N _MOD 2=0 -> INT(MULTIPLY LEFT
                              (EXPAND('**",(%+,1,('**",A,2,),),(N-2)/2,)))
                                  A;
     M _MOD 2=1 -> INT(MULTIPLY('**",X,N-1,)
                       (EXPAND('**",(%-,('**",X,2,),1,),(M-1)/2,
                       )))    X;
              INT(MULTIPLY RIGHT
                  (EXPAND('**",(%-,('**",X,2,),1,),M/2,)))  VAR

_AND SPECIALFORM4 S VAR  ||  IS S RATIONAL IN SIN OR COS??
_BE S=() -> _FALSE;
    ATOMIC S -> _FALSE;
    S=(COS,VAR,) | S=(SIN,VAR,) -> _TRUE;
    SPECIALFORM4 (_HD S) VAR  |  SPECIALFORM4 (_TL S ) VAR


_AND APPLY METHODS S VAR  || APPLY THE METHODS OF INTEGRATION SUGGESTED
_BE _LET APPLY METHODS
    _BE METHODS=() -> _FALSE;
        _NEW A=(_HD METHODS) S VAR
        _IN A=_FALSE -> APPLY(_TL METHODS); A  _IN APPLY METHODS
```

```
_AND INTMULT (OP,LEFT,RIGHT,) VAR
_BE _NEW X=()  ||  X HOLDS THE METHODS APPLICABLE
    _IN NOTCONTAINS VAR LEFT -> FACCONST LEFT,;
        NOTCONTAINS VAR (SIMPLIFY(%/,LEFT,DIFF RIGHT VAR,))
        -> DIFFDIV RIGHT,;
        NOTCONTAINS VAR (SIMPLIFY(%/,RIGHT,DIFF LEFT VAR,))
        -> DIFFDIV LEFT,X;
        _NEW X=ATOMIC LEFT -> X;
              _TL _TL LEFT = ()
              -> NOTCONTAINS VAR (SIMPLIFY(%/,RIGHT,DIFF(_HD _TL LEFT)VAR,))
                 ->DIFFDIV(_HD _TL LEFT),X;
                 X;
              X
        _NEW X=ATOMIC RIGHT ->  X;
              _TL _TL RIGHT=()
                -> NOTCONTAINS VAR (SIMPLIFY(%/,LEFT,DIFF(_HD _TL RIGHT)VAR,))
                   -> DIFFDIV(_HD _TL RIGHT),X;
                X;
              X
        _NEW X=TRIGONOMETRIC(OP,LEFT,RIGHT,) -> INTRIG,X; X
          _NEW X= ATOMIC LEFT
                  -> ATOMIC RIGHT -> X;
                   _HD RIGHT='**" -> MULTEXPAND LEFT RIGHT,INTPARTS,X;
                   _HD RIGHT=%+ | _HD RIGHT=%-
                   -> MULTEXPAND LEFT ('**",RIGHT,1,),X;
                   X;
             X
           _IN ATOMIC RIGHT -> ATOMIC LEFT -> X;
                               INTMULT(OP,RIGHT,LEFT,);
                    _NEW X=_TL _TL RIGHT=() -> INTPARTS,X; X
             _NEW X=_NEW A,S,T,=SPECIALFORM(OP,LEFT,RIGHT,) VAR
                          _IN A -> TRIGSUB S T,X; X
              _NEW X=_NEW A,S,=SPECIALFORM2(OP,LEFT,RIGHT,) VAR
                          _IN A -> NEST S,X; X
              _NEW X=_NEW A,S,T,=SPECIALFORM3(OP,LEFT,RIGHT,)VAR
                          _IN A -> RADSUB(S,T,),X; X
    _IN REVERSE X


_AND INTDIV(OP,LEFT,RIGHT,)VAR  || INTEGRATION METHODS FOR DIVISION
_BE NOTCONTAINS VAR LEFT & LEFT¬=1       -> FACCONST LEFT,;
    NOTCONTAINS VAR RIGHT       -> FACCONST (%/,1,RIGHT,),;
    NOTCONTAINS VAR      (SIMPLIFY(%/,LEFT,DIFF(%/,1,RIGHT,)VAR,))
    ->     DIFFDIV (%/,1,RIGHT,),;
```

```
1009                NOTCONTAINS VAR      (SIMPLIFY(%/,RIGHT,DIFF LEFT VAR,))
1010                ->    DIFFDIV LEFT,;
1011                NOTCONTAINS VAR (SIMPLIFY(%/,LEFT,DIFF RIGHT VAR,))
1012                -> DIFFDIV RIGHT,;
1013                _NEW X=()
1014                _IN _NEW X=ATOMIC LEFT -> X;
1015                             _TL _TL LEFT = ()
1016                             -> NOTCONTAINS VAR(SIMPLIFY(%/,RIGHT,DIFF(_HD_TL LEFT)VAR,))
1017                                -> DIFFDIV(_HD _TL LEFT),X;
1018                                X;
1019                             _HD LEFT='**"
1020                              -> ATOMIC(_HD _TL LEFT)-> X;
1021                              _HD _HD _TL LEFT=SEC |_HD _HD _TL LEFT=CSC
1022                              -> _HD _TL _HD _TL LEFT¬= VAR
1023                              -> DIFFDIV(_HD _TL _HD _TL LEFT),X;
1024                              X;
1025                          X;
1026                     X
1027           _NEW X=
1028                     _NEW A,B,C,=POLYNOMIAL RIGHT VAR
1029                     _IN ¬A -> X;
1030                         B>2 -> X;
1031                         B=2 -> _HD _TL C=0 & _HD _TL _TL C=0 ->X;
1032                         COMPSQUARE C RIGHT,X;
1033              X
1034                _NEW X=_NEW A,S,L,=POLYNOMIAL LEFT VAR
1035                       _AND B,T,M,=POLYNOMIAL RIGHT VAR
1036                       _IN A & B -> S<T -> X; POLYDIV (S,L,) (T,M,), X; X
1037           _NEW X=
1038           _HD RIGHT=%+ | _HD RIGHT=%- ->
1039                _NEW A=INT LEFT VAR
1040                _IN A=_FALSE -> X;
1041                     _NEW B=ARG A VAR
1042                     _IN _TL _TL B¬=() -> X;
1043                         _NEW P,Q,R,=POLYNOMIAL RIGHT B
1044                         _IN P -> B=VAR -> X;
1045                                  DIFFDIV A,X;
1046           X; X
1047                _NEW X=ATOMIC LEFT -> X;
1048                       _HD LEFT=%+ | _HD LEFT=%- -> SEPARATE,X; X
1049                _NEW X=
1050                 _HD RIGHT='**" ->
1051                  _HD _TL _TL RIGHT=(%/,1,2,) ->
1052                   _NEW A=INT LEFT VAR
```

```
1453                    _IN A=_FALSE -> X;
1454                        _NEW  A=ARG A VAR
1455                        _NEW P,Q,R,=POLYNOMIAL A VAR
1456                        _IN ¬P -> X;
1457                            A¬=VAR -> DIFFDIV A,X;
1458                            Q=2 -> COMPSQUARE R (_HD _TL RIGHT) , X;
1459                    X; X; X
1460                _NEW X=TRIGONOMETRIC(OP,LEFT,RIGHT,) -> INTPARTS,INTRIG,X; X
1461                _NEW X=_NEW A,S,T,=SPECIALFORM (OP,LEFT,RIGHT,) VAR
1462                        _IN A -> TRIGSUB S T,X; X
1463                _NEW X=_NEW A,S,T,=SPECIALFORM3 (OP,LEFT,RIGHT,) VAR
1464                        _IN ¬A -> X; RADSUB(S,T,),X
1465                _NEW X=SPECIALFORM4 (OP,LEFT,RIGHT,) VAR -> ARCSUB,X;X
1466                _IN REVERSE X
1467
1468
1469
1470            _AND INTEXP(OP,LEFT,RIGHT,) VAR   ||  METHODS OF INTEGRATING EXPON.
1471            _BE _NEW A,B,C,=POLYNOMIAL LEFT VAR
1472                _IN B=2 -> COMPSQUARE C LEFT,;
1473                    B=1 & LEFT¬=VAR -> DIFFDIV LEFT,;
1474                _NEW X=()
1475                _NEW X=
1476                TRIGONOMETRIC LEFT ->
1477                _HD LEFT= SEC  | _HD LEFT= CSC  ->
1478                _NEW A,B,C,=POLYNOMIAL(_HD _TL LEFT)VAR
1479                _IN ¬A -> _FALSE;
1480                    B=1 -> _HD _TL LEFT¬=VAR ->
1481                            DIFFDIV(_HD _TL LEFT), ;
1482                    RIGHT=2 -> X;
1483                    TRIGEXP,X; X; TRIGEXP,X; X
1484                _NEW X=NOTCONTAINS VAR LEFT      -> DIFFDIV RIGHT,X; X
1485                _NEW X=_NEW A,S,T,=SPECIALFORM (OP,LEFT,RIGHT,) VAR
1486                        _IN A ->  TRIGSUB S T,X; X
1487                _NEW X=_HD LEFT='**" ->
1488                        ATOMIC(_HD _TL LEFT) -> X;
1489                        _NUMBER(_HD _TL _TL LEFT) &
1490                            (_HD _HD _TL LEFT='+ | _HD _HD _TL LEFT='-) ->
1491                    EXPANSION LEFT,X;
1492                X;
1493            X
1494                _NEW X=
1495                _HD LEFT='SIN" | _HD LEFT='COS" |
1496                _HD LEFT='TAN" | _HD LEFT='COT" ->
```

```
1497        _NEW A,B,C,=POLYNOMIAL(_HD _TL LEFT)VAR
1498        _IN ¬A -> X;
1499            B=1 -> _HD _TL LEFT¬=VAR -> DIFFDIV(_HD _TL LEFT),;
1500            X; TRIGEXP, X; X
1501        _NEW X=_NEW A,S,=SPECIALFORM2 (OP,LEFT,RIGHT,) VAR
1502                _IN ¬A -> X; NEST S,X
1503        _NEW X=_NEW A,S,T,=SPECIALFORM3 (OP,LEFT,RIGHT,) VAR
1504                _IN ¬A -> X; RADSUB(S,T,),X
1505        _IN REVERSE X
1506
1507
1508    _AND INTFUNC(FUNC,ARG,)VAR    || INTEGRATION METHODS FOR MONADIC FUNCTIONS
1509    _BE _NEW X=()
1510        _NEW X=
1511        _NEW A,B,C,=POLYNOMIAL ARG VAR
1512        _IN ¬A -> X;
1513            B=1 -> ARG=VAR -> X; DIFFDIV ARG,;    X
1514        _NEW X=_NEW A,S,=SPECIALFORM2(FUNC,ARG,) VAR
1515                _IN ¬A -> X; NEST S,X
1516        _NEW X=_NEW A,S,T,=SPECIALFORM3 (FUNC,ARG,) VAR
1517                _IN ¬A -> X; RADSUB (S,T,),X
1518        _IN REVERSE (INTPARTS,X)
1519
1520
1521    _AND INT S VAR
1522    _BE _NEW S=SIMPLIFY S
1523        _NEW A=BASICFORM S VAR
1524        _IN A¬=_FALSE -> A;
1525            _NEW S=ATOMIC VAR -> S;
1526                    _LET H=_HD VAR
1527                    _IN H=SIN | H=COS | H=TAN |
1528                        H=COT | H=SEC | H=CSC -> CONVERT VAR S;
1529                        S
1530            _IN _HD S=%* -> APPLY(INTMULT S VAR) S VAR;
1531                _HD S=%/ -> APPLY(INTDIV S VAR) S VAR;
1532                _HD S=%+ | _HD S=%- ->
1533                  _NEW (A,B,)=INT(_HD _TL S)VAR, INT(_HD _TL _TL S)VAR,
1534                  _IN  A=_FALSE | B=_FALSE -> _FALSE; _HD S,A,B,;
1535                _HD S=%_ -> _NEW A=INT(_HD _TL S) VAR
1536                              _IN A=_FALSE -> _FALSE; %_,A,;
1537                _HD S="**" -> APPLY(INTEXP S VAR) S VAR;
1538                _TL _TL S=() -> APPLY(INTFUNC S VAR) S VAR;
1539            _FALSE
1540
```

```
1541
1542
1543
1544        ||
1545        ||
1546        ||      PARSER AND INPUT/OUTPUT ROUTINES
1547        ||
1548        ||
1549
1550
1551        _LET OUTPUT S  ||  CONVERTS POLISH TREE TO INFIX NOTATION
1552        _BE ATOMIC S -> S;
1553            OUTOP(_HD S) (_TL S)
1554        _AND OUTOP OP
1555        _BE OP=%+ | OP=%- -> _LAMBDA(LEFT,RIGHT,).
1556             OUTPUT LEFT,OP,OUTPUT RIGHT,;
1557            OP=%_ -> _LAMBDA(OPERAND,).
1558                     ATOMIC OPERAND -> %-,OPERAND,;
1559                     %-,%(,OUTPUT OPERAND,%),;
1560            OP=%* | OP=%/ -> _LAMBDA(LEFT,RIGHT,).
1561                             _NEW LEFT=ATOMIC LEFT -> LEFT;
1562                             OP=%/ -> %(,OUTPUT LEFT,%),;
1563                             _HD LEFT=%+ | _HD LEFT=%- -> %(,OUTPUT LEFT,%),;
1564                               OUTPUT LEFT
1565                             _AND RIGHT=ATOMIC RIGHT -> RIGHT;
1566                             OP=%/ -> %(,OUTPUT RIGHT,%),;
1567                               _HD RIGHT=%+ | _HD RIGHT=%- -> %(,OUTPUT RIGHT,%),;
1568                                   OUTPUT RIGHT
1569                             _IN LEFT,OP,RIGHT,;
1570            OP='**" -> _LAMBDA(LEFT,RIGHT,).
1571              _NEW LEFT=ATOMIC LEFT -> LEFT; %(,OUTPUT LEFT,%),
1572              _AND RIGHT=ATOMIC RIGHT -> RIGHT;
1573                       %(,OUTPUT RIGHT,%),
1574              _IN LEFT,OP,RIGHT,;
1575            _LAMBDA(OPERAND,). ATOMIC OPERAND -> OP,% ,OPERAND,;
1576                               OP,%(,OUTPUT OPERAND,%),
1577
1578
1579        _NEW FAIL=_FALSE,(),()
1580
1581        _LET NEXT S  || THROWS AWAY SPACES IN A LINE OF TEXT
1582        _BE S=() -> ();
1583            _NEW (H,T)=S
1584            _IN H=%  -> NEXT T; H,NEXT T
```

```
1585
1586        _LET EXP S  || FORMS POLISH TREE FROM INFIX DATA IN S
1587        _BE _NEW A,X,S=TERM S
1588            _IN ¬A -> A,X,S;
1589                TERMS X S
1590
1591        _AND TERMS X S  || EXTRA LEVEL OF RECURSION TO HANDLE
1592                        || LEFT RECURSION OF ADDITIONS
1593        _BE S=() -> _TRUE,X,();
1594            _HD S=%  -> TERMS X (_TL S);
1595            ¬(_HD S=%+ | _HD S=%-) -> _TRUE,X,S;
1596            _NEW B,Y,T=TERM(_TL S)
1597            _IN B -> TERMS (_HD S,X,Y,) T;
1598                _FALSE,' NO TERM FOUND FOLLOWING ADDOP ",T
1599
1600        _AND TERM S  || SEARCHING FOR A MULTOP
1601        _BE _NEW A,X,S=FACTOR S
1602            _IN ¬A -> A,X,S;
1603                _NEW S= S=() -> S; _HD S=%  -> NEXT S; S
1604                _IN S=() -> A,X,S;
1605                    ¬(_HD S=%* | _HD S=%/) -> A,X,S;
1606                    _NEW B,Y,T=TERM(_TL S)
1607                    _IN B -> B,(_HD S,X,Y,),T;
1608                        _FALSE,' NO TERM FOUND FOLLOWING MULTOP",()
1609
1610        _AND FACTOR S
1611        _BE _NEW A,X,S=NEGATIVE S
1612            _IN ¬A -> A,X,S;
1613                _NEW S= S=() -> S; _HD S=%   -> NEXT S; S
1614                _IN S=() -> A,X,S; _TL S=() -> A,X,S;
1615                    ¬(_HD S=%* & _HD _TL S=%*) -> A,X,S;
1616                    _NEW B,Y,T=FACTOR(_TL _TL S)
1617                    _IN B -> B,('**",X,Y,),T;
1618                        _FALSE,' NO FACTOR FOUND AFTER EXPONENTIATION ",()
1619
1620        _AND NEGATIVE S  || SEARCH FOR A NEGATIVE SIGN
1621        _BE S=() -> FAIL;
1622            _HD S=%  -> NEGATIVE(_TL S);
1623            _HD S=%- -> _NEW A,X,S=NEGATIVE(_TL S)
1624                        _IN ¬A -> _FALSE,' NO ARG FOUND FOR -VE SIGN",();
1625                             A,(%_,X,),S;
1626            ARG S
1627
1628        _AND ARG S  || THE BOTTOM LEVEL OF THE PARSE
```

```
_BE S=() -> FAIL;
    _HD S=% -> ARG(_TL S);
    _HD S=%( -> _NEW A,X,S=EXP(_TL S)
                _IN S=() -> _FALSE,' MISSING RIGHT PARENTHESIS",();
                    ¬(_HD S=%)) -> _FALSE,' MISSING RIGHT PARENTHESIS",();
                    A,X,_TL S;
_NEW A,X,T=FNEXP S
_IN A -> A,X,T;
    NUMBER S

_AND NUMBER S ||FIND A NUMBER
_BE _LET DIGITS S T
    _BE S=() -> T,S;
        _NEW(A,B)=S
        _IN ¬(_DIGIT A) -> T,S;
            DIGITS(_TL S)(_HD S,T)
    _AND MAKENUM X S
    _BE S=() -> X;
        _NEW X=10*X+(_DIGITVAL(_HD S))
        _IN MAKENUM X (_TL S)
    _IN ¬(_DIGIT(_HD S)) -> FAIL;
        _NEW X,S=DIGITS S ()
        _NEW X=REVERSE X
        _IN _TRUE,MAKENUM 0 X,S

_AND NAME S  || SEARCH FOR NAME OF VARIABLE
_BE _NEW A,X,S=LETTER S
    _IN A -> _NEW B,Y,T=NAME S
             _IN ¬B -> _TRUE,X,S;
                 B,(APPEND X Y),T;
         A,X,S

_AND LETTER S
_BE S=() -> FAIL;
    _LETTER _HD S -> _TRUE,(_HD S,),_TL S;
    FAIL

_AND FNEXP S || LOOK FOR A FUNCTIONAL EXPRESSION
_BE _NEW A,X,S=NAME S
    _IN ¬A -> A,X,S;
        S=() -> A,X,S;
        _NEW B,Y,T=ARG S
        _IN B -> B,(X,Y,),T;  || WAS JUST A NAME ON ITS OWN
            A,X,S
```

```
1673
1674
1675
1676
1677        _NEW DIFFERENTIATE EXPR VAR
1678        _BE _NEW A,X,S=EXP EXPR
1679            _IN ¬A -> 'SYNTAX ERROR: ",X,S;
1680                    'THE DERIVATIVE OF ",_NL,EXPR,_NL,
1681                'WITH RESPECT TO ",VAR, ' IS",_NL,
1682                OUTPUT (SIMPLIFY(DIFF X VAR)),
1683        _AND INTEGRATE EXPR VAR
1684        _BE _NEW A,X,S=EXP EXPR
1685            _IN ¬A -> 'SYNTAX ERROR: ",X,S;
1686                _NEW ANS=INT X VAR
1687                _IN ANS=_FALSE -> 'I AM STUCK";
1688                        'THE INTEGRAL OF",_NL,EXPR,_NL,'IN ",VAR,' IS",_NL,
1689                     OUTPUT(SIMPLIFY ANS),
1690
1691        @LIST     _IN
```

APPENDIX B

SASL BNF

SASL SYNTAX

Expressions

⟨program⟩ : : = ⟨exp⟩
⟨exp⟩ : : = ⟨block⟩ | ⟨λ-exp⟩ | ⟨conditional-exp⟩ | ⟨exp-1⟩
⟨block⟩ : : = let ⟨defs⟩ in ⟨exp⟩
⟨λ-exp⟩ : : = λ⟨formal⟩.⟨exp⟩
⟨conditional-exp⟩ : : = ⟨exp-2⟩→⟨exp⟩;⟨exp⟩
⟨exp-1⟩ : : = ⟨exp-2⟩,⟨exp-1⟩ | ⟨exp-2⟩, | ⟨exp-2⟩
⟨exp-2⟩ : : = ⟨exp-2⟩⟨or-op⟩⟨exp-3⟩ | ⟨exp-3⟩
⟨exp-3⟩ : : = ⟨exp-3⟩ & ⟨exp-4⟩ | ⟨exp-4⟩
⟨exp-4⟩ : : = ⟨¬exp-4⟩ | ⟨exp-5⟩⟨rel-op⟩⟨exp-5⟩ | ⟨exp-5⟩
⟨exp-5⟩ : : = ⟨exp-5⟩⟨add-op⟩⟨exp-6⟩ | ⟨exp-6⟩
⟨exp-6⟩ : : = ⟨add-op⟩⟨exp-6⟩ | ⟨exp-6⟩⟨mult-op⟩⟨exp-7⟩ | ⟨exp-7⟩
⟨exp-7⟩ : : = ⟨ex-op⟩⟨exp-7⟩ | ⟨combination⟩
⟨combination⟩ : : = ⟨combination⟩⟨arg⟩ | ⟨arg⟩
⟨arg⟩ : : = ⟨name⟩ | ⟨constant⟩ | (⟨exp⟩)

Definitions

⟨defs⟩ : : = ⟨def⟩ | ⟨def⟩ and ⟨defs⟩
⟨def⟩ : : = ⟨namelist⟩ = ⟨exp⟩ | ⟨function-form⟩ be ⟨exp⟩
⟨namelist⟩ : : = ⟨formal⟩ | ⟨formal⟩, | ⟨formal⟩ , ⟨namelist⟩
⟨formal⟩ : : = ⟨name⟩ | '(⟨namelist⟩) | ()
⟨function-form⟩ : : = ⟨name⟩⟨formal⟩ | ⟨function-form⟩⟨formal⟩

Various Operators

⟨or-op⟩ : : = |
⟨rel-op⟩ : : = > | >= | = | ¬= | < = | ≮
⟨add-op⟩ : : = + | -
⟨mult-op⟩ : : = * | / mod
⟨ex-op⟩ : : = hd | tl | number | logical | char | list | function | letter | digit | digitval

LAYOUT AND BASIC SYMBOLS

⟨constant⟩ : : = ⟨numeral⟩ | ⟨logical-const⟩ | ⟨char-const⟩ | ⟨string⟩ | ()
⟨numeral⟩ : : = ⟨digit⟩⟨digit⟩*
⟨logical-const⟩ : : = true | false
⟨char-const⟩ : : = % ⟨any character⟩ | nl — the newline character
⟨string⟩ : : = "⟨any message not containing unmatched quotes⟩"
⟨name⟩ : : = ⟨letter⟩ | ⟨name⟩⟨letter⟩ | ⟨name⟩⟨digit⟩
⟨comment⟩ : : = ‖ any message upto the end of the line
⟨ignorable⟩ : : = ⟨space⟩ | ⟨newline⟩ | ⟨comment⟩

APPENDIX C

IDEA'S TABLE OF DIFFERENTIATIONS

$$\frac{d}{dx}(u \pm v) = \frac{du}{dx} \pm \frac{dv}{dx}$$

$$\frac{d}{dx}(-u) = -\frac{du}{dx}$$

$$\frac{d}{dx}(uv) = v\frac{du}{dx} + u\frac{dv}{dx}$$

$$\frac{d}{dx}\left(\frac{u}{v}\right) = \frac{v\frac{du}{dx} - u\frac{dv}{dx}}{v^2}$$

$$\frac{d}{dx}u^a = au^{a-1}\frac{du}{dx}$$

$$\frac{d}{dx}(\sin u) = (\cos u)\frac{du}{dx}$$

$$\frac{d}{dx}(\cos u) = (-\sin u)\frac{du}{dx}$$

$$\frac{d}{dx}(\tan u) = (\sec^2 u)\frac{du}{dx}$$

$$\frac{d}{dx}(\cot u) = (-\csc^2 u)\frac{du}{dx}$$

$$\frac{d}{dx}(\sec u) = (\sec u \tan u)\frac{du}{dx}$$

$$\frac{d}{dx}(\csc u) = -(\csc u \cot u)\frac{du}{dx}$$

$$\frac{d}{dx}(\arcsin u) = \frac{1}{\sqrt{1 - u^2}}\frac{du}{dx}$$

$$\frac{d}{dx}(\arccos u) = \frac{-1}{\sqrt{1 - u^2}}\frac{du}{dx}$$

$$\frac{d}{dx}(\arctan u) = \frac{1}{1 + u^2}\frac{du}{dx}$$

$$\frac{d}{dx}(\operatorname{arccot} u) = \frac{-1}{1 + u^2}\frac{du}{dx}$$

$$\frac{d}{dx}(\operatorname{arcsec} u) = \frac{1}{u\sqrt{u^2 - 1}}$$

$$\frac{d}{dx}(\operatorname{arccsc} u) = \frac{-1}{u\sqrt{u^2 - 1}} \frac{du}{dx}$$

$$\frac{d}{dx}(\sinh u) = \cosh u \frac{du}{dx}$$

$$\frac{d}{dx}(\cosh u) = \sinh u \frac{du}{dx}$$

$$\frac{d}{dx}(\tanh u) = \operatorname{sech}^2 u \frac{du}{dx}$$

$$\frac{d}{dx}(\coth u) = -\operatorname{csch}^2 u \frac{du}{dx}$$

$$\frac{d}{dx}(\operatorname{sech} u) = -\operatorname{sech} u \tanh u \frac{du}{dx}$$

$$\frac{d}{dx}(\operatorname{csch} u) = -\operatorname{csch} u \coth u \frac{du}{dx}$$

$$\frac{d}{dx}(\operatorname{arcsinh} u) = \frac{1}{\sqrt{1 + u^2}} \frac{du}{dx}$$

$$\frac{d}{dx}(\operatorname{arccosh} u) = \frac{1}{\sqrt{u^2 - 1}} \frac{du}{dx}$$

$$\frac{d}{dx}(\operatorname{arctanh} u) = \frac{d}{dx}(\operatorname{arccosh} u) = \frac{1}{1 - u^2} \frac{du}{dx}$$

$$\frac{d}{dx}(\operatorname{arcsech} u) = \frac{-1}{u\sqrt{1 - u^2}}$$

$$\frac{d}{dx}(\text{arccsch } u) = \frac{-1}{u\ 1+u^2}\ \frac{du}{dx}$$

$$\frac{d}{dx}(\text{expn } u) = \text{expn } u\ \frac{du}{dx}$$

$$\frac{d}{dx}(\ln u) = \frac{1}{u}\ \frac{du}{dx}$$

$$\frac{d}{dx}(\log u) = \frac{1}{u}\ \log e\ \frac{du}{dx}$$

$$\frac{d}{dx}\,a^u = a^u \ln a\ \frac{du}{dx}$$

APPENDIX D

IDEA'S TABLE OF INTEGRATION

(constant of integration ignored)

$$\int a\,dx = ax \qquad \int (u+v)dx = \int u\,dx + \int v\,dx \qquad \int axdx = a\int xdx$$

$$\int \sqrt{a^2x^2 + b^2}\,dx = \frac{1}{a}\left(\frac{ax}{2}\sqrt{a^2x^2 + b^2} + \frac{b^2}{2}\ln(ax + \sqrt{ax^2 + b^2})\right)$$

$$\int \sqrt{ax^2 - b^2}\,dx = \frac{1}{a}\left(\frac{ax}{2}\sqrt{ax^2 - b^2} - \frac{b^2}{2}\ln\left(ax + \sqrt{ax^2 - b^2}\right)\right)$$

$$\int \sqrt{b^2 - ax^2}\,dx = \frac{1}{a}\left(\frac{ax}{2}\sqrt{b^2 - ax^2} + \frac{b^2}{2}\arcsin\frac{ax}{b}\right)$$

$$\int \sec^2 x\,dx = \tan x$$

$$\int \csc^2 x\,dx = -\cot x$$

$$\int a^x\,dx = \frac{a^x}{\ln a}$$

$$\int x^a\,dx = \frac{x^{a+1}}{a+1} \quad \text{unless } a = -1 \text{ then } \ln x$$

$$\int \sec x \tan x = \sec x$$

$$\int \csc x \cot x = -\csc x$$

$$\int \frac{1}{x}\,dx = \ln x$$

$$\int \frac{dx}{a^2x^2 + b^2} = \frac{1}{ab}\arctan\left(\frac{ax}{b}\right)$$

$$\int \frac{dx}{a^2x^2 - b^2} = \frac{1}{2ab}\ln\left(\frac{ax - b}{ax + b}\right)$$

$$\int \frac{dx}{b^2 - a^2x^2} = \frac{1}{2ab} \ln \left(\frac{ax + b}{ax - b}\right)$$

$$\int \frac{dx}{\sqrt{a^2x^2 + b}} = \frac{1}{a} \ln \left(ax + \sqrt{a^2x^2 + b}\right)$$

$$\int \frac{dx}{\sqrt{b^2 - a^2x^2}} = \frac{1}{a} \arcsin \frac{ax}{b}$$

$$\int \frac{dx}{x\sqrt{a^2x^2 - b^2}} = \frac{1}{b} \operatorname{arcsec} \left(\frac{ax}{b}\right)$$

$$\int \sin x \, dx = \cos x$$

$$\int \cos x \, dx = \sin x$$

$$\int \tan x = \ln(\sec x)$$

$$\int \cot x \, dx = \ln(\sin x)$$

$$\int \sec x \, dx = \ln(\sec x + \tan x)$$

$$\int \csc x \, dx = \ln (\csc x - \cot x)$$

$$\int \ln x \, dx = x \ln x - x$$

$$\int \log x \, dx = \log e \, (x \ln x - x)$$

$$\int \operatorname{expn} x \, dx = \operatorname{expn} x$$