AIM 57 14 Q

AD785159

STANFORD ARTIFICIAL INTELLIGENCE
MEMO AI-57
ITP-282

October 5, 1967

1

REDUCE
A USER-ORIENTED INTERACTIVE SYSTEM FOR ALGEBRAIC SIMPLIFICATION

by Anthony C. Hearn

ABSTRACT: This paper describes in outline the structure and use of REDUCE, a program designed for large-scale algebraic computations of interest to applied mathematicians, physicists and engineers. The capabilities of the system include:

1) expansion, ordering and reduction of rational functions of polynomials,
2) symbolic differentiation,
3) substitutions for variables and expressions appearing in other expressions,
4) simplification of symbolic determinants and matrix expressions,
5) tensor and non-commutative algebraic calculations of interest to high energy physicists.

In addition to the symbolic operations of addition, subtraction, multiplication, division, numerical exponentiation and differentiation, it is possible for the user to add new operators and define rules for their simplification. Derivations of these operators may also be defined.

The program is written completely in the language LISP 1.5 and is organized so as to minimize the effort required in transferring from one LISP system to another.

Some particular problems which have arisen in using REDUCE in a time-shared environment are also discussed.

Research sponsored by the Air Force Office of Scientific Research, Office of Aerospace Research, U.S. Air Force, Under AFOSR Contract AF49(638)-1389. Computer time supported by the Stanford Artificial Intelligence Project under the sponsorship of the Advanced Research Project Agency of the Office of the Secretary of Defense (SD-183).

SECTION 1: INTRODUCTION

Many of the day to day problems which confront applied mathematicians involve extensive algebraic or non-numerical calculation. Such problems may range from the evaluation of analytical solutions to complicated differential or integral equations on one hand, to the calculation of coefficients in a power series expansion or the computation of the derivative of a complicated function on the other. The difference between these two classes of problems is obvious; in the former case, no straightforward algorithm exists which will guarantee a solution, and indeed an analytic form for the solution may not even exist. On the other hand, algorithms do exist for the solution of problems such as series expansion and differentiation, and so a correct answer may always be found provided the researcher possesses sufficient time, perserverance and accuracy to carry the more complicated problems through free of error. Many examples of this type of problem may be found in physics and engineering. Calculations of general relativistic effects in planetary motion, structural design calculations, and many of the calculations associated with elementary particle physics experiments at high energy accelerators, to name a few, may demand many man-months or even years of work before a useful and error-free answer can be found, even though the operations involved are quite straightforward.

The REDUCE system which is described here has been designed with the latter type of problem in view. Although originally designed for batch processing computer systems, it has been found to be far more useful when used in an interactive, time-shared environment, and it is this form in particular which will be discussed here. Historically,[1] the system was designed for the special problems which are encountered in elementary particle physics, but the techniques have proved capable of extension so that any problem involving manipulation of large algebraic expressions by known algorithmic methods is readily handled. So far, physicists and design engineers have been the principle users of the system as it exists at Stanford University.

The basic problems which the present system is capable of solving include:

1) expansion, ordering and reduction of rational functions of polynomials,
2) symbolic differentiation,
3) substitutions for variables and expressions appearing in other expressions,
4) simplification of symbolic determinants and matrix expressions,
5) tensor and non-commutative algebraic calculations of interest to high energy physicists.

This is by no means a complete list, and several other special features of the system will be described in the course of the paper.

The REDUCE system has been entirely programmed in LISP 1.5[2] a language designed especially for symbol manipulation. The advantage of using such a language is that it is possible to develop a system which is easily capable of modification or extension and is also relatively machine independent. Thus it has proved possible to use the REDUCE system at several IBM 7090 installations, on the time-shared AN/FSQ-32 of System Development Corporation, and the time-shared PDP-6 of Stanford Artificial Intelligence Project. A system for the IBM 360 series will also be available in the near future.

The plan of this paper is as follows. In Section 2, a description of the basic structure of the system is given, explaining the methods employed in simplifying expressions and representing the simplified results. The actual instructions which the user is given for this simplification are described in Section 3, and some remarks on the use and design of the system for time-shared operation are given finally in Section 4.

SECTION 2: BASIC STRUCTURE OF THE SYSTEM

A REDUCE program consists of a set of functional instructions which are evaluated sequentially by a computer. Examples of such instructions are shown in Fig. 1 and the results of a calculation in Fig. 2. The complete description of the elements of the language and the operation of the system are given in a User's Manual[3] and only a brief resume will be given here. Essentially, the arguments of the functional instructions are expressions, which in turn are sequences of numbers, variables operators and standard delimiters (such as commas and parentheses). Allowed numbers in REDUCE statements follow FORTRAN conventions essentially, and may be real or integer. Variables consist of one to twenty-four alphanumeric characters the first of which must be alphabetic. Operators in the system are of two types; infix operators, which occur between their arguments, such as the standard FORTRAN operators ** * / - + = , and prefix operators, which occur at the head of their arguments as in normal mathematical functions, e.g. LOG (logarithms to base e), DET (the determinant operator) and DF, the differential operator. Thus in Fig. 1 DET ((X+Y,X-Y), (X**2 - 2*Y, 3*X)) represents the 2 × 2 determinant

$$\begin{vmatrix} x+y & x-y \\ x^2-2y & 3x \end{vmatrix}$$

and DF(X**2-3*X,X) the derivative of $x^2 - 3x$ with respect to x.

Although the average user of the REDUCE system need know nothing about LISP, a few basic facts are necessary in order to understand how the system operates. We shall assume here that the reader is at least acquainted with list notation as used in most list-processing systems, but not with all the details of LISP.

The input of an expression to the REDUCE system is normally in a FORTRAN-like form as shown in the examples in Fig. 1. The first action of the system is to convert this expression into a standard LISP prefix form, where all operators are written in prefix form (using verbal forms for infix symbols to avoid confusion) and an operator with its list of arguments becomes a list of that operator and its arguments. Thus the expression X-3*X* COS(Y) for example would become (PLUS X (MINUS (TIMES 3 X (COS Y)))).

The majority of the simplification code in the system has been designed to handle rational functions of polynomials. The concept of simplification of an arbitrary algebraic expression remains one of the most controversial subjects in mathematics, but for the class of problems for which the REDUCE system is designed, basic simplification (or reduction) is first achieved by expansion of the expression by the removal of brackets and application of the multinomial theorem. If we restrict our discussion for the time being to rational functions of multivariable polynomials with integer coefficients, then such a reduction is clear-cut and unambiguous.

The expansion operation reduces an expression to a pair of standard forms which represent the numerator and denominator of the expression respectively. The standard form used has undergone considerable modification since the first system was developed[1] and now is a recursive form similar to that described by Collins.[4] This representation has the advantage of faster manipulation and more compact representation of expressions than was achieved with the distributive form described in Ref. 1.

Specifically, an expression in n variables $f(x_1 \ldots x_n)$ is written as a power series in one variable whose coefficents are themselves functions of n-1 variables. Thus

$$f(x_1, x_2 \ldots x_n) = \sum_{i=1}^{m_1} f_i(x_2 \ldots x_n) x_1^i \qquad (1)$$

The polynomial coefficients are themselves expanded in a like manner, and the representation continued until only integers remain. In Bacchus normal form, using the LISP dotted pair notation the REDUCE standard form is:

$$\langle \text{standard form} \rangle ::= \langle \text{integer} \rangle \mid (\langle \text{standard term} \rangle \cdot \langle \text{standard form} \rangle) \tag{2.1}$$

$$\langle \text{standard term} \rangle ::= (\langle \text{standard power} \rangle \cdot \langle \text{standard form} \rangle) \tag{2.2}$$

$$\langle \text{standard power} \rangle ::= (\langle \text{variable} \rangle \,.\, \langle \text{non-zero positive integer} \rangle) \tag{2.3}$$

Thus a standard term represents one term in the power series Eq. (1), and a standard power represents a variable raised to a positive integer power. Standard powers are stored uniquely, but no attempt is made to compact the storage used by other expressions.

An ordering convention based on the machine location of the variable atoms in core is used to decide the position of a variable in this form. Thus two equal polynomials will have the same standard form.

If fractional powers of variables or expressions are encountered during reduction, then a new variable is created to represent that power, and the user informed, ensuring that no fractional powers remain in the standard form. Likewise, any real numbers encountered are usually converted to the ratio of two integers, unless the user specifies otherwise.

An extension of this basic representation to include other operators is made in a particularly simple (but not the most general) manner. To each operator (infix or prefix) there corresponds a simplification function which is called during reduction of the expression to standard forms. If a new operator is added, its simplification function may transform its arguments in two different ways. First, it may convert the expression completely into operators already in the system, leaving no functions of the new operator for further manipulation. This is true of the simplification functions connected with the basic infix operators because the standard form does not include these operators explicitly. It is also true of an operator such as DET, for example, because the relevant simplification function calculates the appropriate determinant, and the operator DET no longer appears. On the other hand, the simplification process may leave some residual functions of the new operator. For example, if the operator COS is added to represent the cosine of its argument, then COS(X) would remain after simplification (as (COS X) in LISP notation) unless a rule for the reduction of cosines (into exponentials, for example) were introduced. These residual functions are termed kernels and are stored uniquely. Subsequently, the kernel is carried through the calculation as a variable unless special transformations on this operator are introduced at a later stage.

To include kernels in our standard form representation, we simply replace Eq. (2.3) by

$$\langle\text{standard power}\rangle ::= (\langle\text{kernel}\rangle \cdot \langle\text{non zero positive integer}\rangle) \qquad (2.3')$$

and add:

$$\langle\text{kernel}\rangle ::= \langle\text{variable}\rangle \mid (\langle\text{operator}\rangle \cdot \langle\text{simplified list of arguments}\rangle) \qquad (2.4)$$

The simplification functions necessary to deal with a new operator are most conveniently written directly in LISP. However, REDUCE does have provision for the inclusion of simple rules for reduction of new operators, and an example of such a rule is given in Fig. 1b.

The simplification functions associated with the expansion of polynomials and the calculation of derivatives, determinants and matrix operations are all examples of straightforward algorithms which are easy to program in LISP. In particular, the recursive nature of the language allows these algorithms to apply to any size or complexity of expression. There are practical limits however on the size of expressions which the system can handle, as no use of secondary storage devices is made during a calculation. Rules for the differentiation of a new operator may also be added as shown again in Fig. 1b.

Most of the operators used in the representation of problems in high energy physics require further reduction after the original expression has been converted into standard forms. Special routines have been written to deal with the explicit problems which occur in this further simplification. Other problems also arise in this class of calculations because a certain amount of non-commutative algebra is encountered, but this is handled in a way which would generalize if further operators of this type were encountered.

SECTION 3 FUNCTIONAL INSTRUCTIONS

There are at present approximately forty functional instructions available to the REDUCE programmer and described in the User's Manual.[3] Although these instructions perform a variety of tasks, they may be divided roughly into two classes; process instructions (or processes) which perform symbolic operations on their arguments and output results to the user, and declaration instructions which perform a variety of service operations prior to the call of a process instruction, such as setting flags controlling output and setting up replacement tables. Process instructions may also add to replacement tables as a by-product of their calculation.

It is impossible within the scope of this paper to describe each instruction in detail and so this section will concern itself rather with some of the more important facets of the system and the instructions which have been designed to handle them, viz., substitutions, input - output, simplification functions, and adding new instructions.

3.1 Substitutions

An important class of instructions in any algebraic simplification system are those which define substitutions for variables and expressions appearing during the calculation. Ideally, it should be possible to replace every occurrence of a given expression $f(a,b..x,y,..)$ by another expression $g(a,b..x,y..)$, where $a,b..$ stand for fixed and $x,y..$ for arbitrary expressions. Thus to quote an example often used, if the system knows that $\sin^2(x) + \cos^2(x) = 1$, where x is arbitrary, then an occurrence of $\sin^2(3a+2\cos(y))+\cos^2(3a+2\cos(y))$, possibly in an expanded form, should be replaced by 1. This general matching problem, at least to the author's knowledge, has not been solved in a manner efficient enough to be used in large scale calculations, and most systems therefore compromise at some point in the types of substitutions allowed.

From the discussion in the previous section, it is readily seen that substitutions can be efficiently implemented for variables, powers of variables, kernels and powers of kernels, all of which are stored uniquely. Substitutions can be made before, during and after reduction to standard forms, and the system decides to some extent the most effective point at which to make the substitution. This decision is also under the user's control, although experience has shown that the system is as good at making a wise decision as the general user. The main point is that the size of intermediate expressions formed during calculation must be kept under control, and so substitutions for expressions which involve a lot of terms in their expansion are usually best made as late as possible.

In addition to those substitutions mentioned above, the system also permits substitutions for products of explicit kernels, or expressions which reduce to this form. Furthermore, any operator which has an explicit replacement in terms of functions of other operators is considered as a variable whose replacement is a functional expression. An example of such a replacement occurs as an argument of the instruction MAKE in Fig. 1a.

One basic ambiguity in the general substitution problem involves the distinction between explicit substitution of the relevant expression and global substitution wherever the expression can be separated out. For example, given that $\sin^2(x) + \cos^2(x) = 1$, should $2\cos^2(a)+\sin^2(a)$ be replaced by $1+\cos^2(a)$, $2-\sin^2(a)$ or left unchanged? The choice made can often influence the compactness or symmetry structure of the result. A good example of this is given in Fig. 3 which shows two expressions which are exactly equivalent, but differ significantly in size and symmetry. This example will be discussed further in the next section. REDUCE does make a distinction, as far as powers of variables and kernels are concerned, between a general substitution and a substitution for the explicit power. For example, LET I**2 = -1 implies that I**3 = -I and I**4 = 1 and so on (LET is an instruction for substitutions in standard forms), while MATCH I**2 = -1 would have no effect on other powers. Examples of these instructions are also given in Fig. 1.

3.2 Input-Output

A considerable number of instructions are available to help the user achieve clarity in output from the system. It has often been pointed out that the standard input and output devices for computers such as key-punches, typewriters and printing devices, are inadequate for representation of mathematical expressions in a form most nearly equivalent to standard mathematical notation. However, experience has shown that the REDUCE expression format using FORTRAN symbols is readily understandable by most engineers and physicists, provided that a certain amount of spacing is used in the display. One very simple aid to clarity consists of putting a number of spaces on each side of an infix operator inversely proportional to the precedence of that operator. 'Factoring' of an expression by separating each power of a given variable or variables, and listing one term to each line, also improve readibility in many cases. The latter format is very convenient if the output is read out onto a secondary storage device, edited by hand, and then read back into the computer for further processing. In this manner, the computer becomes a mathematical scratch-pad, and is an important application of the system when used in a time-shared environment as discussed in the next section.

Quite often, also, a problem requires extensive numerical calculation after a certain amount of non- umerical processing has been undertaken. REDUCE may also be used for this purpose, but as arithmetical operations are rather slow, facilities are provided for producing output in the form of a FORTRAN source program.

3.3 Simplification Instructions

The next class of instructions which we shall briefly consider here are those connected with the 'simplification' of expressions. The principal instruction of this type is SIMPLIFY (or SM) whose main purpose is to reduce its argument by expansion and collection of terms to a pair of standard forms, performing any substitutions which have been declared prior to its call, storing the answer for later use if needed and then printing the results. Examples of the use of this instruction are shown in Fig. 1. As an aid to time-shared operation, the results of a call of SIMPLIFY are always kept until the next call in case the user decides on seeing the results to process them further, or to store them for later use. Most of the other instructions concerned with simplification are used for this further processing of output from SIMPLIFY.

3.4 Adding New Instructions

One disadvantage of an interactive system is that it is necessarily sequential in operation, and therefore the idea of program loops and transfers to earlier statements is rather alien to this concept. However, there are many circumstances in which a series of operations is repeated several times, and to facilitate this, provision is made in REDUCE for the user to write his own instructions in a format essentially the same as an ALGOL 60 procedure and then call for the evaluation of this instruction. This facility is very limited at present, but the next version of the system will include most of the jump and loop statements available in ALGOL. For example, the code in Fig. 1c computes the first twenty Legendre polynomials for any desired argument.

SECTION 4. ON-LINE OPERATION

It should be apparent from our discussion so far that an algebraic manipulation system achieves its greatest effectiveness if used in a highly interactive man-machine environment. The steps which the user takes in solving a problem very often depend on the results of preceeding calculations and so a complete 'program' in the conventional sense is often impossible to write _a priori_. However, interactive use of the REDUCE system poses some problems which are not encountered in batch processed operation. First is the need to provide sufficient safeguards so that the user cannot destroy partial results by making a mistake in input at some point in the calculation. Ideally, the results of all steps in the calculation should be kept, but this would impose impossible burdens on the system if very large expressions are handled. As a compromise measure, REDUCE has been constructed so that all assignments of variables, replacements and rules etc. cannot be changed by a catastrophic error in a subsequent calculation (unless, of course, the user explicitly asks for their removal by mistake), and this has proved an adequate safeguard so far.

Secondly, the standard teletypewriter terminal which most time-sharing systems employ has proved quite inadequate as an output device for the very large expressions often encountered in the problems handled by the system. An obvious solution is the use of CRT displays with suitable editing programs, but little progress has been made in providing such devices cheap enough for general use. A partial solution is to ask the system to print portions of the output, or to store the output on tape or disc and inspect it with an editing program. Alternatively the whole output could be printed on a line printer at a central installation for later collection, but in the latter case much advantage of the interactive operation is lost.

However, these are minor problems compared with the great advantages which time-shared operation offers to the user. An example of the effective use of such a system is given in Fig. 3. Figure 3a is the result of a calculation produced directly by the computer. It contains eleven variables, of which only six are independent. The relations between these variables are shown in Fig. 3b. Figure 3c shows an equivalent expression produced after several hours of hand manipulation and further reduction in the computer. This is not simply a matter of expressing all variables in terms of an independent set, because it can be seen from the compactness and symmetry of the final expression that it was better to use more than six variables. To guess the correct transformations for this reduction is something of an art and it is difficult to see how to program this in a form suitable for the computer. Presumably a beginning step must be to program the computer for factorization of polynomials or computation of the greatest common divisor of two expressions. Considerable progress in these areas has been made over the last few years, and REDUCE does in fact have the latest greatest common divisor algorithm of Collins[4] available to the user if desired. However, the algorithm is still inadequate for efficient handling of the very large expressions encountered in many problems, and so a lot more work will be necessary in this area before a practical routine can be developed.

The method used in obtaining the expression in Fig. 3c from 3a was to use the computer as a scratch pad for checking the accuracy of hand manipulations or the effect of substitutions on small portions of the expression. It is very important that this type of reduction is possible, because an 'answer' consisting of hundreds or thousands of terms provides little insight into the structure of a problem. Unless compact results are achievable, one has to raise the question whether the whole calculation might have been better performed numerically. Quite often the final result is a curve which predicts or compares with experimental conditions, and the algebraic reduction is used as a means of gaining insight into the factors determining that curve. Interactive man-machine computation of the required expressions may prove an important factor in obtaining this insight.

FIGURE CAPTIONS

Figure 1: Typical segments of a REDUCE program. a) Expansion of an expression in order to examine the coefficients of the various powers of x and y.

b) Calculation of a second order partial derivative with respect to x and y of an expression involving operators not already in system.

c) Calculation of the first twenty Legendre polynomials of x^2-2x.

Figure 2: Output from calculation of example in Fig. 1(b).

Figure 3: Example of using REDUCE in an interactive mode. a) Expression initially produced by computer.

b) Relations between variables.

c) Final result produced by hand and machine.

REFERENCES

1. A. C. Hearn, Bull. Am. Phys. Soc., Series 11, 9, 436 (1964), Comm. of the A.C.M. 9, 573 (1966).
2. J. McCarthy et al., LISP 1.5 Programmer's Manual. Computation Center and Research Lab of Electronics, MIT, Cambridge, Mass., 1960.
3. A. C. Hearn, REDUCE User's Manual, Institute of Theoretical Physics Stanford ITP-247, Stanford Artificial Intelligence Memo No. 50, February, 1967 (unpublished).
4. G. E. Collins, Comm. of the A.C.M. 9, 578 (1966) Journal of the A.C.M. 14, 128 (1967).

```
FACTOR X, Y $

MAKE F1(X,W,Z) = X**2+Y**2+W**2-Z**2 ,

     X1 = F1(X,S,L), X2 = F1(X-M,SP,LP) $

SM F = R**2*RP**2*X1*X2 + 2*M*R**2*RP*SP*((X-M) * COSAP - Y*SINAP) *X1

       - 2*M*R*RP**2*DF (DET ((X+Y,X-Y),(X**2-2*Y,3*X)),X)*

          DF(3*X**2-3*X,X) $

LET RP = X+M $ X = 0, XP = 4, LP**2 = 2 $ MATCH L**2*M**2 = (X+Y)**2 $

SM F $
```

(a)

```
RULE SIN (-X) = -SIN(X), COS(-X) = COS(X), SIN(X+Y) = SIN(X)*COS(Y) +

     COS(X)*SIN(Y), COS(X+Y) = COS(X)*COS(Y)-SIN(X)*SIN(Y) $

DERIV DF(SIN(X),X) - COS(X),DF(COS(X),X) = -SIN(Z) $

SIMPLIFY DF(SIN(X-3*X*COS(Y)),X,Y) $
```

(b)

```
INSTRUCTION LEGENDRE X$ BEGIN
INTEGER M; VARIABLE V1,V2,V; LABEL L1$
SET M-2$
SET V1=1$ SET V2 = X$
L1: SET V = (2-1/M)*X*V2-(1-1/M)*V1$
SET M = M+1$
SET V1 - V2$ SET V2 = V$
IF M = 11 THEN FLOATIT ELSE IF M - 20 THEN RETURN
ELSE GO TO L1$
END
LEGENDRE X**2 - 2*X$
```

(c)

FIGURE 1

```
          LISTIT $
    *

          RULE SIN(-X)= -SIN(X), COS(-X)= COS(X),
               SIN(X+Y)= SIN(X)*COS(Y)+COS(X)*SIN(Y),
               COS(X+Y)= COS(X)*COS(Y)-SIN(X)*SIN(Y)$
    *

          DERIV DF(SIN(X),X)= COS(X), DF(COS(X),X)= -SIN(X)$
    *

          SIMPLIFY DF(SIN(X-3*X*COS(Y)),X,Y) $

          9. * X * SIN(X) * COS(Y) * COS(3. * X * COS(Y)) * SIN(Y)
      -  3. * X * SIN(X) * COS(3. * X * COS(Y)) * SIN(Y)
      -  9. * X * COS(Y) * COS(X) * SIN(3. * X * COS(Y)) * SIN(Y)
      +  3. * X * COS(X) * SIN(3. * X * COS(Y)) * SIN(Y)
      +  3. * SIN(X) * SIN(3. * X * COS(Y)) * SIN(Y)
      +  3. * COS(3. * X * COS(Y)) * COS(X) * SIN(Y)

   *
```

FIGURE 2

```
        M**2. *

(2. * RS**2. * RT * QR   +   2. * RS**2. * RT * PR   -   2. * RS**2.
* PT * QR   +   2. * RS**2. * QT * PR   +   2. * RS * RT**2. * QR   +
  2. * RS * RT**2. * PR   +   2. * RS * RT * PS * PR   +   2. * RS * R
T * PT * PR   -   2. * RS * RT * QS * QR   -   2. * RS * RT * QT * QR
  +   2. * RS * PS * QR * PR   +   2. * RS * PT * QR**2.   +   2. * RS
 * PT * QR * PROP1   +   2. * RS * PT * QR * PROP2   +   2. * RS * QS
* QR * PR   +   2. * RS * QT * PR**2.   -   2. * RS * QT * PR * PROP1
  -   2. * RS * QT * PR * PROP2   +   2. * RS * QR**2. * PR   +   2. *
 RS * QR * PR**2.   -   2. * RT**2. * PS * QR   +   2. * RT**2. * QS *
 PR   -   2. * RT * PS * QR**2.   -   2. * RT * PS * QR * PROP1   -
2. * RT * PS * QR * PROP2   -   2. * RT * PT * QR * PR   -   2. * RT *
 QS * PR**2.   +   2. * RT * QS * PR * PROP1   +   2. * RT * QS * PR *
 PROP2   -   2. * RT * QT * QR * PR   +   2. * RT * QR**2. * PR   +
2. * RT * QR * PR**2.)


   -   2. * RS**2. * PT * QT * QR   -   2. * RS**2. * PT * QT * PR
+   2. * RS * RT * PS * QT * QR   -   2. * RS * RT * PS * QT * PR   -
  4. * RS * RT * PS * QR * PR   +   RS * RT * PS * QR * PROP1   -   2.
 * RS * RT * PT * QS * QR   +   2. * RS * RT * PT * QS * PR   -   4. *
 RS * RT * PT * QR * PR   +   RS * RT * PT * QR * PROP1   -   4. * RS
* RT * QS * QR * PR   -   RS * RT * QS * PR * PROP1   -   4. * RS * RT
 * QT * QR * PR   -   RS * RT * QT * PR * PROP1   +   4. * RS * PS * P
T * QT * QR   +   2. * RS * PS * PT * QR**2.   -   4. * RS * PS * QT**
2. * PR   -   2. * RS * PS * QT * QR * PR   +   4. * RS * PT**2. * QS
* QR   -   4. * RS * PT * QS * QT * PR   +   2. * RS * PT * QS * QR *
PR   -   RS * PT * QR * PR * PROP2   -   2. * RS * QS * QT * PR**2.
-   RS * QT * QR * PR * PROP2   +   2. * RT**2. * PS * QS * QR   +   2
. * RT**2. * PS * QS * PR   -   4. * RT * PS**2. * QT * QR   -   4. *
RT * PS * PT * QS * QR   +   2. * RT * PS * PT * QR**2.   +   4. * RT
* PS * QS * QT * PR   +   2. * RT * PS * QT * QR * PR   +   RT * PS *
QR * PR * PROP2   +   4. * RT * PT * QS**2. * PR   -   2. * RT * PT *
QS * QR * PR   -   2. * RT * QS * QT * PR**2.   +   RT * QS * QR * PR
* PROP2) / (


2. * RS * RT * QR * PR * PROP1 * PROP2)


*
```

(a)

FIGURE 3

```
PROP1 = 2*PR - 2*PS + 2*PT

PROP2 = PROP1 + 2*RS - 2*RT

   QR = PR + RS - RT

   QS = PT + PR - RT

   QT = PS - PR - RS
```

(b)

```
((2. * (PS   +   QT)**2.   +   2. * (PT   +   QS)**2.   +   (RS   +
 RT)**2.   +   (PR   +   QR)**2.) * ((RS * PR   +   RT * QR) * (PS   +
   QT)   -   (RS * QR   +   RT * PR) * (PT   +   QS))   -   8. * M**2.
 * ((PS   +   QT)**2. * (RS * PR   +   RT * QR)   +   (PT   +   QS)**2
. * (RS * QR   +   RT * PR)   -   (RT   -   TS) * (PR * QR   -   RT *
RS) * (PS   +   PT   +   QS   +   QT)   -   (PS   +   QT) * (PT   +
QS) * (RS   +   RT) * (PR   +   QR)   +   (PR * QR   +   RS * RT) * (R
S   +   RT) * (PR   +   QR))) / (   -   8. * PROP1 * PROP2 * PR * QR *
 RS * RT)
```

(c)

FIGURE 3