S-1103

STANFORD ARTIFICIAL INTELLIGENCE PROJECT
Memo No. 50

February 1967

Institute of Theoretical Physics
ITP-247

REDUCE USERS' MANUAL

by Anthony C. Hearn

Abstract: REDUCE is a program designed for general algebraic computations of interest to physicists and engineers. Its capabilities include:

1) expansion and ordering of rational functions of polynomials,

2) symbolic differentiation,

3) substitutions in a wide variety of forms,

4) reduction of quotients of polynomials by cancellation of common factors,

5) calculation of symbolic determinants,

6) calculations of interest to high energy physicists including spin 1/2 and spin 1 algebra.

The program is written completely in the language LISP 1.5 and may therefore be run with little modification on any computer possessing a LISP 1.5 compiler or interpreter.

Research sponsored by the Air Force Office of Scientific Research, Office of Aerospace Research, U.S. Air Force, under AFCSR Contract AF 49(638)-1389. Computer time supported by the Advanced Research Project Agency of the Office of the Secretary of Defense (SD-183).

a
+
Stanf univ
LISP 1.5.

TABLE OF CONTENTS

TABLE OF CONTENTS (cont.)

SECTION 1. INTRODUCTION

REDUCE is a program designed for general algebraic computations of interest to physicists and engineers. Its capabilities include:

1) expansion and ordering of rational functions of polynomials,
2) symbolic differentiation,
3) substitutions in a wide variety of forms,
4) reduction of quotients of polynomials by cancellation of common factors,
5) calculation of symbolic determinants,
6) calculations of interest to high energy physicists including spin 1/2 and spin 1 algebra.

The program is written completely in the language LISP 1.5(1) and may therefore be run with little modification on any computer possessing a LISP 1.5 compiler or interpreter.

Versions of the program have operated at several batch-processing IBM 7090 installations, on the time-shared AN/FSQ-32 of System Development Corporation, and the time-shared PDP-6 of the Stanford Artificial Intelligence Project. This report is intended primarily for users of the system on the latter machine (referred to as REDUCE2) and on the Stanford IBM 7090 (REDUCE1).

There are three levels at which REDUCE may be used and understood.

(1) For calculations using only the functions already in REDUCE. This requires no knowledge of LISP or the details of the REDUCE program. This operation, which will be adequate for most users, is described in Section 2.

(2) To develop and use new functions written in terms of the primitives of the REDUCE system. This requires a knowledge of LISP, but little knowledge

of the details of the REDUCE program, and is described in Part II of this manual.

(3) To modify and develop the primitives of REDUCE, which requires a complete knowledge of LISP and REDUCE. A description of the system for this purpose will be published elsewhere.

Section 3 contains programs and output of sample calculations which have been run on the Stanford PDP-6. Finally a summary of instructions available and a list of possible diagnostic messages and reserved variables is given in Section 4.

Most sections contain details of a certain amount of material of interest only to high-energy physicists. Knowledge of this is not necessary for successful operation of the system. Sub-sections dealing only with this material will be starred, and may be omitted by those not interested.

REDUCE is part of a larger system designed for semi-automatic calculations involving Feynman diagrams in quantum electrodynamics and particle physics, and described briefly in Reference (2). Those parts of the full system dealing with Feynman graph generation and manipulation will also be described in other publications.

The author would appreciate hearing from any users who experience trouble with the system (please include copies of relevant input and output). Acknowledgement of the use of REDUCE in any published calculations would also be appreciated.

SECTION 2.

2.1 Preliminary

A REDUCE program consists of a set of functional instructions which are evaluated sequentially by a computer. Examples of such instructions, which are explained in this Section are:

```
MAKE  X = Y + 2, Z(W) = W**2/7;
LET  EPS(P,Q,R,S) = 0 $
PUNCHIT  $
SIMPLIFY  (X**2 - Y**2)/(X - Y) $
```

A program is terminated by the instruction

```
END $
```

The arguments of these functions are expressions which in turn are sequences of numbers, variables, operators and standard delimiters (such as commas and parentheses). The allowed form for these elements is as follows.

2.2 Numbers

Numbers in REDUCE statements may be of two types; integer and real.

Integers consist of a signed or unsigned sequence of 1-11 decimal digits written without a decimal point.

e.g. -2, 5396 +32

Real numbers may be written in two ways;

i) as a signed or unsigned sequence of 1-9 decimal digits with an embedded or terminating decimal point, but not beginning with a decimal point;

ii) as in i) followed by a decimal exponent which is written as the letter E followed by a signed or unsigned integer.

e.g. 32.

+32

0.32E2

320.E-1

are all representations of 32.

Not allowed: .5

-.52E3

The system normally uses integer arithmetic which is required by the greatest common divisor algorithm. Under standard running conditions, a real number is converted into the ratio of two integers. A message will also be printed to indicate the conversion,

e.g. 3.4 REPRESENTED AS 17/5.

In REDUCE2 it is possible to operate using real arithmetic, in which case no check for greatest common divisors will be made. The declaration FLOATIT should be used if this mode is required, while NOFLOAT returns the system to integer arithmetic.

A distinct disadvantage of the present system is that single precision arithmetic only is available. It is hoped that provision for multiple or arbitrary precision arithmetic will be included in REDUCE2 in the near future.

N.B. In REDUCE1, any real number within 10^{-6} of an integer is converted automatically to that integer, and numbers with absolute value less than 10^{-6} are converted to zero. These restrictions do not apply to REDUCE2.

2.3 Variables

Variables in REDUCE are specified by name and type. There are two types; scalar and vector.

Variable names consist of one to twenty-four alphanumeric characters (i.e. alphabetic letters or numbers) the first of which must be alphabetic.

e.g. A, AZ, P1, Q23P,

AVERYLONGVARIABLE234

Type specification is implicit for scalar variables, but must be explicit for vectors. One way to do this is to use the declaration VECTOR.

e.g. VECTOR P1, Q23P $

specifies that P1 and Q23P are vector variables. The instructions INDEX (Section 2.4.2) and MASS (Section 2.7.14) also declare their arguments to be vectors.

2.3.1 Reserved variables

Several scalar and vector variables in REDUCE have a particular value which cannot be changed by the user. These reserved variables should therefore be used only for the purpose intended. For example, the scalar variable I is used to represent $\sqrt{-1}$, and all occurrences of I**2 will be replaced by -1. A list of reserved variables is given in Section 4.2.

2.4 Operators

Operators in the REDUCE system are also specified by name and type. There are two types, infix and prefix.

Infix operators occur between their arguments.

e.g. A + B - C, B**2/C, (P.Q)

Such operators in the system are

.	scalar product of two four vectors (see Section 2.4.2*)	binary
**	exponentation	binary
/	division	binary
*	multiplication	n-ary
-	subtraction (or unary minus)	binary or unary
+	addition (or unary plus)	n-ary
=	equivalence	binary

Parentheses may be used to specify the order of operation. If parentheses are omitted, then the order of combination is by the precedence ordering given by the above list (from innermost operations to outermost operations).

<u>Prefix</u> operators occur at the head of their arguments, which are written as a list enclosed in parentheses and separated by commas, as in normal mathematical functions.

```
e.g.  LOG (X)
      DET ((X,Y), (Y,X))
      DF (X,X)
      G (L,P,Q)
```

In REDUCE1, it is also possible to use the Stanford Burroughs B5500 ALGOL character set for operators. Thus the following operators are considered equivalent.

*	×	⟨68 punch⟩
=	←	⟨058 punch⟩

The character ; ⟨-68 punch⟩ may also be used instead of $.

However, in order that the system can recognize `*` as the exponentiation operator when reading ALGOL, the user must use the command BMODE NIL (before the BEGIN card as explained in the job setup instructions) to affect this. In REDUCE2, the symbol ↑ may also be used to represent exponentiation.

Prefix operators in the system are

DET	denotes determinant	n-ary
DF	partial differentiation of first argument with respect to remaining arguments	n-ary
LOG	logarithm to base e	unary
G	gamma matrix expression	n-ary
EPS	completely antisymmetric tensor of degree four	quaternary

These operators and the . operator are described below.

2.4.1 Special Operators

(a) DET

The operator DET is used to represent $n \times n$ determinants. DET has n arguments interpreted as rows of the determinant each of which is a list of n expressions. For example the determinant

$$\begin{vmatrix} A & B & C \\ D & E & F \\ G & H & J \end{vmatrix}$$

would be written

```
DET ((A,B,C), (D,E,F), (G,H,J))
```

N.B. If the determinant is larger than 8×8, the present routines (which expand recursively in terms of minors of the first row) become prohibitively slow.

(b) DF

The operator DF is used to represent partial differentiation with respect to one or more variables. The first argument is the scalar expression to be differentiated and the remaining are the differentiation variables, the order of the variables specifying the order of differentiation

e.g. $\mathrm{DF}(E,X) \equiv \frac{\partial E}{\partial X}$

$\mathrm{DF}(E,X,Y) \equiv \frac{\partial}{\partial Y}\left(\frac{\partial E}{\partial X}\right)$

$\mathrm{DF}(E,X,X) \equiv \frac{\partial^2 E}{\partial X^2}$

etc.

where E is any scalar expression.

If substitutions (Section 2.7.3) have been declared for any variables in E, then these substitutions are checked for dependence on the differentiation variables.

(c) LOG

LOG is used to represent logarithms to base e, and is a function of one argument, which is a scalar expression. Little effort is made by the system to simplify LOG expressions. They are differentiated correctly, but no attempt at combination or expansion is made.

2.4.2* Operators Used in High-Energy Physics

(a) .

The . operator is a binary operator used to denote the scalar product of two Lorentz four-vectors. In the present system, the index handling routines all assume that Lorentz four-vectors are used, but these routines could be rewritten to handle other cases.

Components of vectors can be represented by including representations of unit vectors in the system. Thus if EO represents the unit vector (1,0,0,0), (P·EO) represents P_o, the zeroth component of the four-vector P. Our metric and notation follows Bjorken and Drell.(3) Similarly, an arbitrary component P_μ may be represented by (P.U). If contraction over components of vectors is required then the instruction INDEX must be used. Thus

```
        INDEX  U  $
```

declares U as an index, and the simplification of

```
        (P·U) * (Q·U)
```

would result in

```
        (P·Q)
```

Arguments of INDEX are also flagged as vectors.

The metric tensor $g_{\mu\nu}$ may be represented by (U·V). If contraction over μ and ν is required, then U and V should be declared as indices.

During the index contraction phase, the system checks to see that all indices declared are both matched and used in every term. If not, a terminal error message results. If the user wishes to declare more indices than occur in every term, the instruction IFLAG will turn off the check for redundant indices, but not the check for unmatched indices.

The instruction REMIND V1...VN $ may be used to remove the index (and vector) flags from the variables V1 through VN.

(b) G

G is an n-ary operator used to denote a product of gamma matrices contracted with Lorentz four-vectors. Gamma matrices are associated with fermion lines in a Feynman diagram. If more than one such line occurs, then a different set of gamma matrices (operating in independent spin spaces) is required to represent each line. To facilitate this, the first argument of G is a line identification variable (not a number) used to distinguish different lines.

Thus

G(L1,P) * G(L2,Q)

denotes the product of $\not{p}$ associated with a fermion line identified as L1, and $\not{q}$ associated with another line identified as L2 and where P and Q are Lorentz four-vectors. A product of gamma matrices associated with the same line may be written in a contracted form.

Thus

G(L1, P1, P2..., P3) ≡ G(L1, P1) * G(L1, P2)*...*G(L1, P3)

The vector A is reserved in arguments of G to denote the special gamma matrix γ_5 .

Thus

G(L,A) ≡ γ_5 associated with line L

G(L,P,A) ≡ $\not{p} * \gamma_5$ associated with line L.

γ_μ (associated with line L) may be written as G(L,U), with U flagged as an index if contraction over μ is required. The notation of Bjorken and Drell$^{(3)}$ is assumed in all operations involving gamma matrices.

(c) EPS

The operator EPS has four arguments, and is used only to denote the completely antisymmetric tensor of order 4 and its contraction with Lorentz four-vectors

Thus

$$\epsilon_{\mu\nu\rho\sigma} = \begin{cases} +1 & \text{if } \mu,\nu,\rho,\sigma \text{ is an even permutation of } 0,1,2,3. \\ -1 & \text{if an odd permutation} \\ 0 & \text{otherwise} \end{cases}$$

A contraction of the form $\epsilon_{\mu\nu\rho\sigma}p^{\rho}q^{\sigma}$ may be written as EPS(U,V,P,Q), and so on.

2.5 Expressions

REDUCE expressions may be of four types; scalar, vector, matrix and equivalence and consist of syntatically allowed sequences of numbers, variables, operators, left and right parentheses and commas.

2.5.1 A scalar expression follows the normal rules of algebra subject to the following restrictions:

(1) numerical exponents only are allowed in expressions. Furthermore, only integer exponents are permitted in the standard representation of expressions. Again, this restriction is required by the greatest common divisor routines. Conversion of expressions with real exponents to the required form is made by the system and a message is printed to inform the user of this.

Examples of scalar expressions are:

```
X
X**3 -2*Y/(2*Z**2 - DF(X,Z))
(P**2 + M**2)**(1/2)*LOG(Y/M)
(2.5*X - Y/1.2)**1.2
```

2.5.2* Vector Expressions follow the normal rules of vector combination. Thus the product of a scalar expression and a vector expression is a vector expression, as are the sum and difference of vector expressions. If these rules are not followed, error messages occur indicating either the absence of a vector variable, or the presence of too many vector variables in an expression. Assuming P and Q have been declared vectors, the following are vector expressions

```
P
P -2*Q
2*X*Y*P - (P.Q)*Q/(3*Q.Q)
```

whereas P*Q and P/Q are not.

2.5.3* Matrix Expressions denote those expressions involving gamma matrices. A gamma matrix is a 4×4 matrix, and so the product, sum and difference of such expressions is again a matrix expression. There are no matrix variables in the system, and wherever a scalar variable appears in a matrix expression without an associated gamma matrix, an implicit unit 4×4 matrix is assumed.

e.g. G(L,P) + M denotes G(L,P) + M*⟨unit 4×4 matrix⟩.

N.B. multiplication of matrix expressions is of course non-commutative.

2.5.4 Equivalence expressions contain the equivalence operator, = . Their general form is

⟨scalar vector or matrix expression⟩ = ⟨scalar vector or matrix expression⟩

for example

```
X   = 4 - Z/2

P   = M

A*B = 4  .
```

The equivalence operator is binary, and so an expression of the form

```
A = B = C
```

is not allowed.

2.6 Kernels

A particular type of expression of great importance in the REDUCE system is a kernel. It may be defined as one of the structures

⟨variable⟩ ⟨operator⟩ ⟨variable⟩   for infix operators

or   ⟨operator⟩(⟨variable⟩,....⟨variable⟩)for prefix operators

where   ⟨operator⟩ is one of the operators ** . G or EPS .

In cases where the arguments of these operators may be reordered, the system puts the kernel arguments in a canonical order, based on the intrinsic order of the variables (Section 2.7.2) and stores the kernels uniquely. We therefore define a kernel form as an expression of the form given above, whose arguments are not necessarily in the canonical order.

Examples of kernel forms are:

```
A**2

P.Q

G(L,P,Q)
```

whereas

```
A*B

(A+B)**2

EPS(P,Q+R,S,T)
```

are not.

2.7 Functional Instructions

Functional instructions are instructions to the computer to perform some operation. They consist of an instruction name, a list of arguments (which may be empty), separated by commas, and an instruction terminator, $ or ; . Nearly all functional instructions are described in this Section, but the user should consult Section 4.1 for a complete list.

Functional instructions 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 declaring variable types, setting flags controlling output and setting up replacement tables. Process instructions may also add to replacement tables as a by-product of their calculation.

We shall illustrate the use of these instructions by considering first the process SIMPLIFY.

2.7.1 SIMPLIFY (or SM)

The argument of SIMPLIFY is a scalar or matrix expression. The main purpose of SIMPLIFY is to reduce this argument by expansion and collection of terms to a quotient of two standard polynomial forms. The standard form used by the system is similar in structure to that of G. E. Collins.(4) In addition, a standard ordering of variables is used in expressions, and this may be specified by the user. During this reduction, various types of substitutions may be made for variables and kernels in the expression. In addition, derivatives, determinants, contractions of indices and traces of gamma matrices are calculated if required. The result of these operations is then printed and stored for later use if needed.

Roughly, the operations of SIMPLIFY or its argument follow the following sequence:

(1) Substitutions of the first kind (described in Section 2.7.4).

(2) Conversion to quotient of two standard polynomial forms, including calculation of derivatives and determinants.

(3) Index contraction and traces of γ-matrix expressions if required.

(4) Substitutions of second kind (Section 2.7.5).

(5) Cancellation of greatest common divisor, if required (Section 2.7.8).

(6) Output of results (Section 2.7.9).

A large number of declarations may be used in connection with SIMPLIFY and most other processes. For example the instruction TITLE takes a single argument which appears as a title on process output. Another simple example is the instruction for ordering variables in expressions.

2.7.2 <u>Ordering of variables</u> is defined at read-in time, the variable with the highest order being that read first. This order is retained throughout the calculation. All variables in expressions are ordered in terms of their intrinsic order, and the speed of a calculation and the size of expressions can depend on this order. For this reason it is wise to give variables which occur most frequently the highest order.

The instruction ORDER may be used to order variables, although the position of variables as they are read in also determines their order. Thus

```
ORDER  X, Y, Z $
```

orders X ahead of Y, Y ahead of Z and all ahead of other variables in expressions which follow. ORDER should be the first instruction in a calculation (unless FACTOR is also used [Section 2.7.9]), otherwise variables introduced in earlier instructions will be ordered ahead of those in the ORDER declaration.

Reserved variables (Section 5.2) already have an intrinsic order in the system, and this cannot be changed by the user. In general, their order is lower than any variable introduced by the user.

2.7.3 Substitutions

An important class of instructions are those which define substitutions on variables and expressions in the argument of SIMPLIFY. These fall naturally into two classes; substitutions on general expressions (substitutions of the first kind) defined by the instruction MAKE and substitutions on standard forms and quotients (substitutions of the second kind) defined by the instruction LET.

2.7.4 Substitutions of First Kind

These substitutions are declared by the instruction MAKE. The argument of MAKE is a list of equivalence expressions of the form

```
          (1)   ⟨variable⟩  =  ⟨expression⟩

                     e.g.  X  =  Y**2 + 2
                           C  =  G(L,Q) + M
```

or

```
          (2)   ⟨variable⟩(⟨variable⟩,⟨variable⟩...)  =  ⟨expression⟩
                     e.g. F(U)    = U + 3
                          H(U,V) = G(L,U,V) -G(L,V,U)
```

In case (1), all occurrences of the variable on the left of the equivalence sign are replaced by the expression on the right. Case (2) defines a functional substitution. All occurrences of the functional name are considered as a function with the declared number of arguments and the appropriate substitutions. If the number of arguments do not match an error occurs. For example, with the above substitutions the expression

```
          X**2 + 2*X*F(Y+Z)
```

becomes

```
          (Y**2+2)**2+2*(Y**2+2)*((Y+Z)+3)
```

If the left hand side of an equivalence expression is redefined by a later call of the instruction, the previous expression is replaced by the new one, and a diagnostic message printed to inform the user.

The instruction

```
CLEAR  V1...VN $
```

may be used to remove the variables V1 through VN from the replacement tables. In the case of functional definitions only the functional name should appear in the arguments of CLEAR. If any of the variables V1 through VN are not found, a diagnostic message is printed.

2.7.5 Substitutions of the Second Kind

These substitutions, which define replacements in standard forms, are declared by the instruction LET. The argument of LET is a list of equivalence expressions of the form:

```
(1)  ⟨variable⟩     =  ⟨expression⟩
(2)  ⟨kernel form⟩  =  ⟨expression⟩
```

Examples are

```
X      = Y + Z
P1     = Q - 2*M*R/(M1+M2)
(P.R)  = (S - M**2)/2
Y**3   = 2*Z - 3
```

The implementation of these substitutions is very efficient, as they are defined in terms of kernels which are stored uniquely.

In most cases, the instruction MAKE is sufficient for defining replacements for scalar variables. However, if the instruction RSM is following, LET must then be used, as explained in Section (2.7.10). In addition, LET should be used in cases where it is obviously more efficient to make the substitution after reduction to standard forms rather than before.

Substitutions of the form (2) allow additions of real exponents to the system in a convenient manner. For example, suppose the expression

```
(P**2 + M**2)**0.5      (a)
```

is required in a calculation. By setting

```
X**2 = P**2 + M**2      (b)
```

then X can be used to represent the root, and the system will replace all even powers of X by the appropriate number of powers of P**2 + M**2. Any derivatives with respect to variables in such statements are made correctly. If an expression of the form (a) is encountered during simplification, it is automatically replaced by a new variable and a substitution of the form (b) generated. A message is also printed to inform the user of this

e.g. (P**2 + M**2)**(1/4) REPRESENTED BY G0123

The remarks on redefining equivalence expressions and the instruction CLEAR in Section 2.7.4 also apply to LET.

2.7.6 Asympotic Constraints

In expansions of polynomials involving variables which are known to be small, it is often desirable to throw away all powers of these variables beyond a certain point to avoid excessive unnecessary computation. The instruction LET may be used conveniently to do this. For example, if only powers of x up to x^7 are needed, the instruction

```
LET X**8 = 0 $
```

will inform the system to keep the required terms and delete all others.

2.7.7 Limitations in use of MAKE and LET

There are several features of these instructions of which the user should be aware.

First, no variable on the left of a replacement expression may appear in the right of the same expression.

Thus

```
X = X + Z, Y = Y
```

would be incorrect arguments of MAKE or LET.

Secondly, a check is made at the end of every instruction call for variables or functions in the righthand side of each expression which are themselves replaced in another substitution of the same kind. Thus a call of

```
        MAKE  X = Y + Z,   Z = L + M $
```

would result in the X replacement being stored as

```
        X = Y + L + M
```

If Z were redefined by a further call of MAKE, the replacement for X would change accordingly. However a call of the instruction CLEAR Z $ would <u>not</u> change the definition X, and a subsequent definition of Z would have no effect on X.

As a consequence however of the checking facility of MAKE and LET, any <u>implicit</u> substitution of a variable in terms of itself is not allowed. Thus

```
        MAKE   L = M + N,  N = L + R $
```

is illegal.

It should be noted that MAKE and LET replacements are kept entirely separate in the system and no checking for common substitutions is made between them.

Lastly, there are several key variables which cannot appear in the left half of substitutions. If one of these is used, a diagnostic message will be printed stating that the replacement was not allowed. For example, system prefix operators cannot appear in the left half of equivalence expressions.

2.7.8 Cancellation of Common Factors

Facilities are available in REDUCE for cancelling common factors in the numerators and denominators of expressions, at the option of the user. If required the system computes the greatest common divisor of the numerator and denominator using an algorithm due to G. E. Collins(5) and cancels this divisor from the relevant terms. Unfortunately, large integers outside the single-precision range of the present system are often generated and will result in a terminal error condition if encountered. It is hoped to remedy this by introducing multiple or arbitrary precision arithmetic in the near future. The instruction FACIT causes the system to check for common factors, while NOFAC returns the system to its normal state.

A check is automatically made, however, for common kernels in the denominators of expressions. These are divided into the numerator, which may result in negative exponents appearing in printouts of results, even though only positive powers are kept in standard forms.

2.7.9 Output of Expressions

A considerable amount of effort has been devoted in REDUCE to the printing of expressions in the most convenient and readable form. For example, infix operators are set off by spaces, the number of spaces being (inversely) dependent on the precedence of the operator; thus ** has no spaces each side whereas = has four. The standard form of output of an expression is as a list of terms, single spaced and filling the whole print line. However, the user has at his disposal a wide range of declarations which modify the printing, none of which need be used if not required.

These are:

(a) FACTOR. This instruction takes a list of scalar variables as argument. FACTOR is not really a factoring command, but rather a separation command. All terms involving fixed powers of the declared variables are printed as a product of the fixed powers and a sum of the rest of the terms. An example of such factorization is shown in Section 3.2. In order for the relevant algorithm to operate efficiently variables being factored should have highest order. Thus the FACTOR command should be the first command in the program (preceeding even an ORDER command) for efficient operation. The instruction REMFAC V1,..VN $ removes the factoring flag from the variables V1 through VN.

(b) LISTIT. Often the output is easier to handle if each term is printed on a separate line. The declaration LISTIT achieves this, and may be turned off by using NOLIST.

(c) SPACEIT. This instruction with no arguments may be used to double space output in REDUCE1 only. NOSPACE, similarly, returns printing to the normal form.

(d) PUNCHIT and PFORT are punching instructions (with no arguments) available in REDUCE1 only. Punched output is designed for use as source program in numerical calculations. The former instruction punches expressions close-packed in ALGOL notation (compatible with Burroughs B5500 input), whereas the latter punches FORTRAN IV-compatible output. Cards punched by PUNCHIT may also be used as input in REDUCE calculations provided ALGOL input has been declared (Section 2.4). Punching may be discontinued by using NOPUNCH.

2.7.10 Further Manipulation of SIMPLIFY Output may be achieved in a variety of ways.

First, the results of all process calculations are saved as a quotient of two standard polynomial forms which may be further reduced by the instruction RSM. The user can make further substitutions (using LET), and change factorization and output conditions before "resimplifying" the result. New substitutions of the first kind (using MAKE) will have no effect as all reductions are made on standard forms.

After the expression has been "resimplified" its new value is stored in the system and so the "resimplification" process may be continued indefinitely.

Secondly, the argument of SIMPLIFY only (not other processes) may be an equivalence expression of the form

(1) ⟨variable⟩ = ⟨expression⟩

or

(2) ⟨variable⟩ (⟨variable⟩,...⟨variable⟩) = ⟨expression⟩

as in arguments of MAKE. In this case the expression on the right of the equivalence is simplified as before and stored as a substitution of the first kind with the left of the equivalence. It may then be used in further process calculations.

Thus it is possible to have the same argument for MAKE, LET and SIMPLIFY, but the effect is entirely different. Consider, for example

```
MAKE   X  =  DF (Y**2, Y) $
LET    X  =  DF (Y**2, Y) $
SM     X  =  DF (Y**2, Y) $
```

In the first and second cases, the replacement would be stored as it is on different tables to use as a substitution on general expressions or standard forms respectively. In the third case, the right hand side would be simplified to give

```
        2*Y
```

and so the system would store the replacement

```
        X  =  2*Y
```

as a substitution of the first kind.

Lastly in REDUCE2 the user can save the result of a simplification by calling the function SAVEAS after the relevant call of SIMPLIFY. The argument of SAVEAS is a variable or functional form as in the left half of MAKE arguments. The result is then saved as a substitution of the first kind. This command is useful mainly in time-shared operation.

2.7.11 Adding Results of Process Calculations

If the user requires the sum of a series of process calculations the declaration SUMIT will cause the cumulative results of all processes after that call of SUMIT to be saved rather than each result. The results of each process will be printed as usual. A call of RSM will then cause that sum to be reduced and printed. Furthermore, the instruction TOTAL will print the cumulative result without checking for new substitutions. Both RSM and TOTAL turn off SUMIT, so it must be set again if required later. A call of SAVEAS will save the cumulative results of all processes rather than that immediately preceding if SUMIT is active.

2.7.12 Numerical Evaluation of Expressions is also possible in REDUCE by replacing variables and kernel forms with numbers using MAKE and LET. It should be pointed out however that the arithmetic routines in LISP are not very efficient and it is wiser to use results as FORTRAN or ALGOL source decks if extensive arithmetic is required.

2.7.13 Other Processes

There is one other process instruction presently available in REDUCE, but more will be added as the need arises. This instruction is only of interest to high-energy physicists and is described below.

2.7.14* REDUCE

The argument of REDUCE is a matrix expression corresponding to one fermion line. REDUCE converts the expression to a quotient of two standard polynomials, expressing in the process all gamma matrix products in terms of the 16 fundamental gamma matrices.

Furthermore it is possible to left or right anticommute a vector in a gamma expression to the end of the expression, where it will be replaced by its mass (if defined) by assuming an implicit spinor and applying the Dirac equation. Three declarations are available for this, namely

```
MASS    P1  =  M1, P2  =  M2  etc $
```

which assigns a mass M1 to variable P1 etc, and

```
LCOM P,Q...R $
```

and

```
RCOM P,Q,...R $
```

which declare left and right anticommutation for the relevant variables in products of gamma matrix expressions respectively and which must come after the mass declaration. If any of these fermions are antiparticles, the declaration

```
ANTIPTL P,Q...R $
```

will ensure that the relevant vector is replaced by the negative of its mass.

The instruction REDUCE is not normally part of the system but is available on request.

2.8 Spacing

In general, spaces in input programs are ignored by REDUCE except where they are required to avoid confusion. The user may therefore use spaces to set out his program as he wishes subject to the following simple restrictions:

(i) spaces may NOT occur between the individual letters of variable names;

(ii) a space MUST occur after the instruction name in a functional instruction;

(iii) a space MUST occur after the $ or ; at the end of the instruction.

In REDUCE1, columns 1 through 72 may be used for program. In addition, all card boundaries and, in REDUCE2, file boundaries, carriage returns, line and form feeds and tabs are ignored and CANNOT be used in lieu of necessary spaces as defined above.

SECTION 3. EXAMPLES

In this Section we give simple examples of REDUCE programs of increasing complexity. These programs have all been run on the Stanford PDP-6 and results in most cases are given. The user can easily check his knowledge of the system by writing his own versions of each problem and running them. The job instructions for the relevant machine (Section 4.4) should be consulted for details of system loading and program input.

All these examples were first set-up as files in the PDP-6 system (roughly the equivalent of a card-deck for the 7090), and the whole file run as explained in the job instructions (Section 4.4.2). Alternatively, each instruction could be typed and the user wait for results before proceeding to the next instruction. An example of this form of operation is given in Section 4.4.2.

Times taken for running these examples are not quoted, but were all of the order of a few seconds or less.

3.1 Differentiation and Determinants

The first program is designed to calculate the following partial derivatives:

1) $\frac{\partial}{\partial x}(x^2 + y^2)$

2) $\frac{\partial^2}{\partial x \partial y}\left(\frac{x^2+y^2}{x^2-y^2}\right)$

3) $\frac{\partial^2}{\partial x \partial y}\frac{\log(x^2+y^2)}{x^2}$

4) $\frac{\partial^2}{\partial x \partial y}\left\{(x+2y)\begin{vmatrix} x^2+y^2 & x^2-y^2 \\ x+y & y \end{vmatrix}\right.$

$\left. + \begin{vmatrix} x^2 & y \\ y^2 & x \end{vmatrix}\right\}$

A possible program would be:

```
 SM DF(X**2+Y**2,X) $
SM DF((X**2+Y**2)/(X**2-Y**2),X,Y) $
SM DF(LOG(X**2+Y**2)/X**2,X,Y)$
SM DF((X+2*Y)*DET((X**2+Y**2, X**2-Y**2) , (X+Y, Y)),X,Y)
    + DF(DET((X**2,Y),(Y**2,X)),X,Y) $
END $
```

where we have split the fourth derivative into two parts for variety. (Remember that SM is an alternative form for SIMPLIFY)

Alternatively, we could make use of the fact that x^2+y^2 and x^2-y^2 occur often to write instead

```
MAKE P= X**2+Y**2, Q= X**2-Y**2$
SM DF(P,X) $ SM DF(P/Q,X,Y) $
SM DF(LOG(P)/X**2,X,Y)$
SM DF((X+2*Y)*DET((P,Q),(X+Y,Y)),X,Y)+DF(DET((X**2,Y),
   (Y**2,X)),X,Y)$
END $
```

The results of running this program are:

```
MAKE   P   =   X**2.   +   Y**2.,      Q   =   X**2.   -   Y**2.;

*

SM      DF(P,X);


2. * X


*

SM      DF(P / Q,X,Y);


(   -   8. * X**5. * Y   +   8. * X * Y**5.) / (X**8.   -   4. * X**6
. * Y**2.   +   6. * X**4. * Y**4.   -   4. * X**2. * Y**6.   +   Y**8
.)


*

SM      DF(LOG(P) / X**2.,X,Y);


(   -   8. * X * Y   -   12. * X**(-1.) * Y**3.   -   4. * X**(-3.) *
 Y**5.) / (X**6.   +   3. * X**4. * Y**2.   +   3. * X**2. * Y**4.   +
   Y**6.)


*

SM      DF((X   +   2. * Y) * DET((P,Q),(X   +   Y,Y)),X,Y)   +   DF(
DET((X**2.,Y),(Y**2.,X)),X,Y);


   -   6. * X**2.   +   4. * X * Y   +   12. * Y**2.


*
```

In calculating the second derivative the system did not cancel a common factor in numerator and denominator. We illustrate in the next program how this could be done

```
FACIT $ SM DF((X**2+Y**2)/(X**2-Y**2),X,Y) $ END $
```

Results of running this program are:

```
FACIT   ;

*

SM      DF((X**2.   +   Y**2.) / (X**2.   -   Y**2.),X,Y);


(   -   8. * X**3. * Y   -   8. * X * Y**3.) / (X**6.   -   3. * X**4
. * Y**2.   +   3. * X**2. * Y**4.   -   Y**6.)


*
```

Alternatively, we could print results before and after cancellation as in the following program

```
SM DF((X**2+Y**2)/(X**2-Y**2),X,Y) $
FACIT $ RSM $ END $
```

3.2 Expansion of Polynomials

This example is part of a larger calculation which arose in a design engineering problem. Our aim is to expand the following expression and examine the coefficients of x and y

$$f = r^2 \times rp^2(x^2+y^2+s^2-\ell^2)((x-m)^2+y^2+sp^2-\ell p^2)$$
$$+ 2m \times r^2 \times rp \times sp((x-m)\ \cos(ap)-y \times \sin\ (ap))$$
$$(x^2+y^2+s^2-\ell^2) - 2m \times r \times rp^2 \times s(x \times \cos(a)-y \times$$
$$\sin\ (a))\ ((x-m)^2+y^2+sp^2-\ell p^2)$$

As no expansion or differentiation of the sines or cosines is required, they can easily be incorporated by expressing each as a variable. Thus we write COSAP for cos(ap) and so on.

The most straightforward way to input the problem is to type in the whole expression, asking also for "factorization" of x and y in order to examine the coefficients more easily, as in the following program

```
FACTOR X,Y $
SIMPLIFY R**2*RP**2*(X**2+Y**2+S**2-L**2)*((X-M)**2+Y**2
  +SP**2-LP**2) +2*M*R**2*RP*SP*((X-M)*COSAP-Y*SINAP)*
  (X**2+Y**2+S**2-L**2)-2*M*R*RP**2*S*(X*COSA-Y*SINA)*
  ((X-M)**2+Y**2+SP**2-LP**2) $
END $
```

However, much less typing in involved if we observe that several terms have the same functional structure as in the next program. In addition, after calculating the required result we have defined substitutions for y, rp, s and sp (using LET, since the substitutions are made on standard forms) and "resimplified" the result. We have also used ↑ instead of ** to denote exponentiation. Note however that the system prints results in the latter form.

```
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 R↑2*RP↑2*X1*X2 + 2*M*R↑2*RP*SP*((X-M)*COSAP-Y*SINAP)*
   X1 - 2*M*R*RP↑2*S*(X*COSA-Y*SINA)*X2 $
LET RP = X+M, S= Ø, SP= Ø, Y= 4 $ RSM $ END $
```

Results of this calculation are:

```
FACTOR  X,      Y;
(Y X)

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      R**2. * RP**2. * X1 * X2   +   2. * M * R**2. * RP * SP * ((X
  -    M) * COSAP   -   Y * SINAP) * X1   -   2. * M * R * RP**2. * S
* (X * COSA   -   Y * SINA) * X2;


        X**4. *
(R**2. * RP**2.)

   +   X**3. *
(  -   2. * S * M * R * RP**2. * COSA   +   2. * M * SP * R**2. * RP
 * COSAP   -   2. * M * R**2. * RP**2.)

   +   X**2. * Y**2. *
(2. * R**2. * RP**2.)
```

```
   +    X**2. * Y *
(2. * S * M * R * RP**2. * SINA    -    2. * M * SP * R**2. * RP * SINA
P)

   +    X**2. *
(S**2. * R**2. * RP**2.    +    4. * S * M**2. * R * RP**2. * COSA    -
  L**2. * R**2. * RP**2.    -    2. * M**2. * SP * R**2. * RP * COSAP
+    M**2. * R**2. * RP**2.    +    SP**2. * R**2. * RP**2.    -    LP**2.
* R**2. * RP**2.)

   +    X * Y**2. *
(   -    2. * S * M * R * RP**2. * COSA    +    2. * M * SP * R**2. * RP
 * COSAP    -    2. * M * R**2. * RP**2.)

   +    X * Y *
(   -    4. * S * M**2. * R * RP**2. * SINA)

   +    X *
(2. * S**2. * M * SP * R**2. * RP * COSAP    -    2. * S**2. * M * R**2
. * RP**2.    -    2. * S * M**3. * R * RP**2. * COSA    -    2. * S * M *
 SP**2. * R * RP**2. * COSA    +    2. * S * M * LP**2. * R * RP**2. * C
OSA    -    2. * L**2. * M * SP * R**2. * RP * COSAP    +    2. * L**2. *
M * R**2. * RP**2.)

   +    Y**4. *
(R**2. * RP**2.)

   +    Y**3. *
(2. * S * M * R * RP**2. * SINA    -    2. * M * SP * R**2. * RP * SINA
P)

   +    Y**2. *
(S**2. * R**2. * RP**2.    -    L**2. * R**2. * RP**2.    -    2. * M**2.
 * SP * R**2. * RP * COSAP    +    M**2. * R**2. * RP**2.    +    SP**2. *
 R**2. * RP**2.    -    LP**2. * R**2. * RP**2.)

   +    Y *
(   -    2. * S**2. * M * SP * R**2. * RP * SINAP    +    2. * S * M**3.
 * R * RP**2. * SINA    +    2. * S * M * SP**2. * R * RP**2. * SINA    -
   2. * S * M * LP**2. * R * RP**2. * SINA    +    2. * L**2. * M * SP *
 R**2. * RP * SINAP)

   -    2. * S**2. * M**2. * SP * R**2. * RP * COSAP    +    S**2. * M**
2. * R**2. * RP**2.    +    S**2. * SP**2. * R**2. * RP**2.    -    S**2.
* LP**2. * R**2. * RP**2.    +    2. * L**2. * M**2. * SP * R**2. * RP *
 COSAP    -    L**2. * M**2. * R**2. * RP**2.    -    L**2. * SP**2. * R**
2. * RP**2.    +    L**2. * LP**2. * R**2. * RP**2.


*
```

```
LET     RP   =   X   +   M,     S   =   Ø.,     SP   =   Ø.,     Y   =
   4.;

*

RSM     ;



     X**6. *
(R**2.)

   +   X**4. *
(   -   L**2. * R**2.    -   2. * M**2. * R**2.    -   LP**2. * R**2.
 +   32. * R**2.)

   +   X**3. *
(   -   2. * M * LP**2. * R**2.    +   32. * M * R**2.)

   +   X**2. *
(2. * L**2. * M**2. * R**2.    +   L**2. * LP**2. * R**2.    -   16. *
L**2. * R**2.    +   M**4. * R**2.    -   M**2. * LP**2. * R**2.    -   1
6. * M**2. * R**2.    -   16. * LP**2. * R**2.    +   256. * R**2.)

   +   X *
(2. * L**2. * M * LP**2. * R**2.    -   32. * L**2. * M * R**2.    -
32. * M * LP**2. * R**2.    +   512. * M * R**2.)

   -   L**2. * M**4. * R**2.    +   L**2. * M**2. * LP**2. * R**2.    -
   16. * L**2. * M**2. * R**2.    +   16. * M**4. * R**2.    -   16. * M
**2. * LP**2. * R**2.    +   256. * M**2. * R**2.


*
```

Alternatively, we could have saved the answer to the first part as the variable F, say, and called SM again after defining the substitutions (using MAKE or LET this time) as follows

```
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*S*(X*COSA-Y*SINA)*X2 $
MAKE RP = X+M, S= Ø, SP= Ø, Y= 4 $ SM F$ END $
```

This calculation would have been a little slower than the previous example, because the system has to convert F to standard forms again.

3.3* Calculation of Lowest Order Compton Scattering Cross-Section

We reproduce here as an example of a simple calculation in high energy physics the computation of the Compton scattering cross-section as given in Bjorken and Drell,(3) Eqs. (7.72) through (7.74).

We wish to compute

$$\frac{\alpha^2}{2}(k'/k)^2 \text{ trace}\left\{\left(\frac{\not p_f+m}{2m}\right)\left(\frac{\not\epsilon'\not\epsilon\not k_i}{2k\cdot p_i}+\frac{\not\epsilon\not\epsilon'\not k_f}{2k'\cdot p_i}\right)\left(\frac{\not p_i+m}{2m}\right)\left(\frac{\not k_i\not\epsilon\not\epsilon'}{2k\cdot p_i}+\frac{\not k_f\not\epsilon'\not\epsilon}{2k'\cdot p_i}\right)\right\}$$

where k_i, k_f are the four-momenta of incoming and outgoing photons (with polarization vectors ϵ and ϵ' and laboratory energies k and k' respectively) and p_i, p_f are incident and final electron four-momenta.

Omitting the factor $\frac{\alpha^2}{8m^2}\left(\frac{k'}{k}\right)^2$ we need to find

$$\mathrm{trace}\left\{ (\not p_f + m) \left(\frac{\not\epsilon' \not\epsilon \not k_i}{2k_i \cdot p_i} + \frac{\not\epsilon \not\epsilon' \not k_f}{2k_f \cdot p_i}\right) (\not p_i + m) \left(\frac{\not k_i \not\epsilon \not\epsilon'}{2k_i \cdot p_i} + \frac{\not k_f \not\epsilon' \not\epsilon}{2k_f \cdot p_i}\right) \right\}$$

The most straightforward REDUCE program for this, with appropriate substitutions would be:

```
LET PI.E= Ø, PI.EP= Ø, PI.PF= M**2+KI.KF, PI.KI= M*K,PI.KF=
    M*KP, PF.E= -KF.E, PF.EP= KI.EP, PF.KI= M*KP, PF.KF=
    M*K, KI.E= Ø, KI.KF= M*(K-KP), KF.EP= Ø, E.E= -1, EP.EP=
    -1, PI.PI= M**2, PF.PF= M**2, KI.KI= Ø, KF.KF= Ø;
TITLE COMPTONCXN ;
SIMPLIFY (G(L,PF)+M)*(G(L,EP,E,KI)/(2*KI.PI) + G(L,E,EP,KF)/
   (2*KF.PI)) * (G(L,PI)+M)*(G(L,KI,EP,E)/(2*KI.PI) +
   G(L,KF,EP,E)/(2*KF.PI)) ;
END $
```

However, we can use the replacement facilities to decrease the amount of input, as in the following program. This calculation will also run faster, too, because the system makes use of the MSHELL information while calculating the necessary traces. (MSHELL is described in Section 4.1)

```
MASS KI= Ø, KF= Ø, PI= M, PF= M;
MSHELL KI,KF,PI,PF;
LET PI.E= Ø, PI.EP= Ø, PI.PF= M**2+KI.KF, PI.KI= M*K,PI.KF=
    M*KP, PF.E= -KF.E, PF.EP= KI.EP, PF.KI= M*KP, PF.KF=
    M*K, KI.E= Ø, KI.KF= M*(K-KP), KF.EP= Ø, E.E= -1, EP.EP=
    -1 ;
MAKE GP(P)= G(L,P)+ M;
TITLE COMPTONCXN;
SM GP(PF)*(G(L,EP,E,KI)/(2*KI.PI) + G(L,E,EP,KF)/(2*KF.PI))
  * GP(PI)*(G(L,KI,E,EP)/(2*KI.PI) + G(L,KF,EP,E)/(2*KF.PI)) ;
END ;
```

Results are:

```
MASS     KI    =   Ø.,     KF    =   Ø.,     PI    =    M,      PF    =    M;

*

MSHELL   KI,       KF,       PI,       PF;

*

LET       PI.E    =    Ø.,   PI.EP    =    Ø., PI.PF    =    M**2.    +   KI.KF
,         PI.KI    =    M * K,      PI.KF    =    M * KP,      PF.E     =
-    KF.E,      PF.EP    =    KI.EP,        PF.KI    =    M * KP,       PF.KF
  =    M * K,    KI.E    =    Ø.,   KI.KF    =    M * (K    -   KP),    KF.EP
  =    Ø.,       E.E    =       -    1.,     EP.EP    =       -    1.;

*
```

```
MAKE    GP(P)    =    G(L,P)    +    M;

*

TITLE   COMPTONCXN;

*

SM      GP(PF) * (G(L,EP,E,KI) / (2. * KI.PI)   +   G(L,E,EP,KF) / (2
. * KF.PI)) * GP(PI) * (G(L,KI,E,EP) / (2. * KI.PI)   +   G(L,KF,EP,E)
/ (2. * KF.PI));


COMPTONCXN    =

2. * K * KP**(-1.)    +    2. * K**(-1.) * KP    +    8. * E.EP**2.    -
 4.


*
```

3.4* Calculation of Basic Traces in Radiative Corrections to Electron Positron Scattering

The program below, which was designed to study the basic traces in radiative corrections to electron positron scattering is presented without comment as an example of a fairly complicated calculation. Results, which cover many pages, are not given.

```
LISTIT ;

VECTOR P,Q,R,S,T,U,V,W,P1,P2,P3,P4,K;

MAKE GP(P)= G(L,P)+M, H(P)= G(L,P);

MASS P= M, Q= M, R= Ø, S= M, T= M;

MSHELL P,Q,R,S,T; LET R.W= Ø;

INDEX W;

MAKE X1(P,Q)= 2*(U.V)+4*((P.U)*(Q.V)+(P.V)*(Q.U))/PROP ;

SM X2(P,Q,R)= -GP(P)*H(U)*GP(Q+R)*H(W)*GP(Q)*H(W)*GP(R+Q)
        *H(V)/(4*(Q.R)↑2*PROP) ;

SM X4(P,Q,R)= GP(Q+R)*H(U)*GP(P)*H(W)*GP(P-R)*H(V)*GP(Q)
        *H(W)/(4*(Q.R)*(P.R)*PROP) ;

PAUSE ; REMIND W;

SM X3(P,Q,R)=GP(P)*H(U)*GP(Q)*H(V)*GP(P+R)*H(W)/(2*(P.R));

SM X31(P,Q,R)=GP(P+R)*H(U)*GP(Q)*H(V)*GP(P)*H(W)/(2*(P.R));

INDEX U,V,W;

SM X5(P,Q,S,T,R)= -GP(P)*H(U)*GP(Q)*H(V)*GP(S)*H(W)*GP(S+R)*
        H(U)*GP(T)*H(W)*GP(T-R)*H(V) *(2*S.R)*(-2*T.R) ;

SM X6(P,Q,S,T,R)= -GP(P)*H(U)*GP(Q)*H(V)*GP(S)*H(W)*
        GP(S+R)*H(U)*GP(T)*H(W)*GP(T-R)*H(V) *(-4*S.R*T.R);

SM X7(P,Q,S,T,R)= - GP(P)*H(U)*GP(Q)*H(V)*GP(S)*H(U)*
        GP(T+R)*H(W)*GP(T)*H(W)*GP(T+R)*H(V) *(-4*S.R*T.R);

PAUSE ; REMIND U,V,W;

SM YA(P,Q,R,S,T)= X1(P,Q)*(X2(S,T,R)+2*X4(S,T,R)+X2(T,S,-R));

SM YB(P,Q,R,S,T)= -2*(X3(P,Q,R)+X3(Q,P,-R))*(X31(S,T,-R)+
        X31(T,S,R)) ;

SM YC(P,Q,R,S,T)= 2*(X5(P,Q,S,T,R)+X6(P,Q,S,T,R)+X6(T,P,Q,S,R)
        + X7(T,P,Q,S,R)) ;

END ;
```

SECTION 4. SUMMARY OF SYSTEM

4.1 Instructions Normally Available in REDUCE

Notation:	E, E1...EN	denote expressions
	U1,...UN	denote lists
	V1,...VN	denote variables

The number after the description refers to the section in which the instruction is described.

BMODE NIL (REDUCE1 only)	declares ALGOL character set for input. (MUST occur before BEGIN card). (2.4)
CLEAR V1..VN $	removes variables V1 through VN from replacement tables. (2.7.4)
CONT $ (REDUCE2 only)	instruction to continue evaluating REDUCE file (see PAUSE).
END $	terminates calculation. (2.1)
FACIT $	instruction to cancel common factors in numerator and denominator of results. (2.7.8)
NOFAC $	turns off cancel instruction. (2.7.8)
FACTOR V1,..VN $	declares a list of variables as factors in the answer. Returns list of current factor variables. (2.7.9)
REMFAC V1..VN $	Removes arguments from list of factoring variables. (2.7.9)
FLOATIT $ (REDUCE2 only)	instruction to use real rather than integer arithmetic. (2.2)
NOFLOAT $ (REDUCE2 only)	instruction to return to integer arithmetic. (2.2)
IFLAG $	turns off error check for unused indices as defined by INDEX. (2.4.2)
INDEX V1,..VN $	declares arguments as indices and flags them as vectors. Returns list of current indices. (2.4.2)

REMIND V1..VN $	removes index (and vector) flags on arguments. (2.4.2)
LET E1,..EN $	declares substitutions of second type. Arguments are equivalence expression of the form: ⟨scalar variable⟩ = ⟨scalar expression⟩, or ⟨vector variable⟩ = ⟨vector expression⟩, or ⟨kernel form⟩ = ⟨expression⟩. (2.7.5)
LISTIT $	declaration to list output one term to a line. (2.7.9)
NOLIST $	turns off above. (2.7.9)
MAKE E1,..EN $	declares substitutions of first type. Arguments are equivalence expressions of the form ⟨variable⟩ = ⟨expression⟩ or ⟨function name⟩ (⟨argl⟩,..⟨argn⟩) = ⟨expression⟩. (2.7.4)
MASS E1,..EN $	Arguments are equivalence expressions of the form ⟨vector variable⟩ = ⟨scalar variable⟩. Where applicable, the scalar variables are now interpreted as masses belonging to the corresponding vector variables. The vector variables are also flagged as vectors in the system. (2.7.14)
MSHELL V1..VN $	Arguments are vector variables. For each variable, is a mass has previously been assigned (by MASS), a "mass-shell" substitution of the second kind of the form, for example (P.P) = M**2 is set up.
ORDER V1..VN $	specifies canonical order for variables. (2.7.2)
PAUSE $ (REDUCE2 only)	this instruction causes the system to stop during the running of a REDUCE file and a message CONT? to appear on the user's teletype. If Y is now typed, the calculation continues. If any other character is typed the user may then input further program from the teletype.

	If the user desires later to return to the point in the file where the PAUSE occurred, the instruction CONT $ achieves this.
PFORT $ (REDUCE1 only)	instruction to punch relevant output in FORTRAN 4-compatible form. (2.7.9)
PUNCHIT $ (REDUCE1 only)	instruction to punch relevant output in ALGOL - compatible form. (2.7.9)
NOPUNCH $ (REDUCE1 only)	turns off punching modes. (2.7.9)
PRINREP $ (REDUCE2 only)	prints current replacement table (of all types of substitutions).
RSM $	re-simplifies output from previous calculation. (2.7.10)
SAVEAS E $ (REDUCE2 only)	saves the result of the last call of a process instruction as a substitution of the first kind. E has the same form as the left half of an argument of MAKE. (2.7.10)
SUMIT $	tells system to add results of each process calculation so that RSM or TOTAL prints cumulative results. (2.7.11)
SPACEIT $ (REDUCE1 only)	declares double-spaced output, unless punching is called for, when it has no effect. (2.7.9)
NOSPACE $ (REDUCE1 only)	turns off double-spacing. (2.7.9)
SIMPLIFY E $ or SM E $	E is an expression which is reduced as explained in Section (2.7.1).
TITLE V $	V is printed at top of output. (2.7.1)
TOTAL $	prints results of previous calculation, or cumulative results if SUMIT declared. (2.7.11)
VECTOR V1,..VN $	declares list of vector variables as vectors. (2.3)

4.2 Reserved Variables

We list here all variables, operators, and instruction names which remain in REDUCE once the BEGIN command has been called. Not all of these are reserved in the strictest sense, as an instruction name for example could be used as a scalar variable without calamity, or a reserved scalar variable could be used as a vector variable and so on. All these names have an intrinsic order in the system which cannot be changed.

Reserved scalar variables:	E (reserved in numbers only), I
Reserved vector variables:	A
Operator names:	DET, DF, EPS, G, LOG
Instruction names:	BEGIN, CLEAR, CONT, END, FACIT, FACTOR, FLOATIT, IFLAG, INDEX, LET, LISTIT, MAKE, MASS, MSHELL, NOFAC, NOFLOAT, NOLIST, NOPUNCH, NOSPACE, ORDER, PAUSE, PFORT, PUNCHIT, PRINREP, REMFAC, REMIND, RSM, SAVEAS, SIMPLIFY, SM, SPACEIT, SUMIT, TITLE, TOTAL, VECTOR.

4.3 REDUCE Diagnostic and Error Messages

Diagnostic messages in the REDUCE system are of two basic types, terminal error messages and warning messages. The former result in the termination of the calculation whereas the latter warn the user of some action by the system which may also indicate possible errors on his part but do not cause the calculation to terminate. A terminal error may also indicate a system error, and for this reason the messages are listed under three headings,

(1) messages indicating terminal errors on the part of the user,
(2) diagnostic warning messages,
(3) messages indicating terminal errors due to system failure.

The messages listed are usually accompanied by a printout of diagnostic information which may be useful to a consultant or a person who understands the working of the system. In particular REDUCE1 error printouts are headed by a message ERROR NUMBER *A1* , indicating an error called by the REDUCE program rather than a LISP system error message.

4.3.1 Terminal Error Messages

DET MISMATCH	length of rows and columns of a determinant are not equal.
NUMERICAL EXPONENTS REQUIRED	exponents must be numbers.
LOG 0 UNDEFINED	zero argument for log operator detected.
MISMATCH OF ARGUMENTS (REDUCE2 only) (In REDUCE1, this error causes system errors F2 or F3)	a functional substitution declared by MAKE occurs with wrong number of arguments in an expression.
MISSING VECTOR	a vector expression lacks a vector.
REDUNDANT VECTOR	a vector expression involves erroneous product of vectors.
READ ERROR } SYNTAX ERROR }	indicate errors in program input.
UNMATCHED OR UNUSED INDICES ⟨list of relevant indices⟩ DECLARED	a check is made to ensure <u>all</u> indices declared by INDEX are used and matched. Provided those indices which are used are matched, it is possible to turn off check for unused indices by using IFLAG. (2.4.2)
0/0 FORMED	system has detected 0/0.
⟨variable⟩ HAS NO MASS	a mass is required for this variable. (May also appear as a diagnostic message.)

The following message indicates that the problem, in its present form, is too large for the system. Often a rearrangement of substitutions will allow the problem to run, and, in any case, a consultant's advice should be sought.

In REDUCE1: ERROR NUMBER *GC2*

In REDUCE2: NO STORAGE LEFT

4.3.2 Diagnostic Messages

NEW REPLACEMENT DEFINED FOR ⟨expression⟩	LET or MAKE has been asked to replace the same expression twice, the new replacement is then used.
READ ERROR MISSING OPERATOR	two expressions occur without an operator in between.
READ ERROR REDUNDANT OPERATOR	two operators occur together.
REPLACEMENT ⟨expression⟩ NOT ALLOWED	LET or MAKE has been given illegal replacement.
UNKNOWN CHARACTER	illegal character occurs in program.
⟨expression⟩ NOT FACTORABLE	only variables may be factored.
⟨expression⟩ REPRESENTED BY ⟨expression⟩	real numbers and rational powers of expressions are converted by the system to other forms.
⟨expression⟩ NOT DEFINED	indicates call of a functional instruction not known to system.
⟨variable⟩ NOT FOUND	indicates that an argument of CLEAR is not on any replacement table.

4.3.3 System Error Messages

The following error messages should not occur during normal running of REDUCE programs. If they do, the program should be discussed with consultant.

ASIGN	ISIMP2	REMOVE
CARX	LISIND	SIMPATOM
EMULT	MULTN*	SIMPDOT
EPS	NLIST	SIMPGAMMA
GCD	PERMP	SIMPTIMES
GCDK	PERM1	VSIMPTIMES
GCD1	PLACE1	⟨expression⟩ HAS NO SUBR

There are also several error messages called by the 7090 LISP system. these have the form

ERROR NUMBER *⟨error reference⟩*

e.g. ERROR NUMBER *I3*

If such errors occur, the output should be shown to a consultant. Error messages called by the PDP-6 LISP system are a little more informative, but should also be shown to a consultant if they occur.

4.4 REDUCE Job Setup

Although the form of a REDUCE program remains the same from machine to machine, the action necessary to run the job naturally varies. This section describes the particular features associated with running jobs at the installations where REDUCE is available.

4.4.1 <u>REDUCE Job Setup for Stanford 7090</u>

REDUCE for the Stanford 7090 is stored in various versions as a core dump on the 1301 Disk. A REDUCE job consists of the following sequence of cards

No. 1 card

No. 2 card

$$\left.\begin{array}{l}\langle\text{REDUCE packet}\rangle\\ \langle\text{REDUCE packet}\rangle\\ \langle\text{REDUCE packet}\rangle\\ \langle\text{REDUCE packet}\rangle\end{array}\right\}\quad\text{one or more}$$

A REDUCE packet consists of the following sequence:

first card: TEST (beginning in column 8)

second card: BEGIN NIL STOP

subsequent cards: ⟨REDUCE instruction⟩ ⟨REDUCE instruction⟩ etc END $

(as in examples in Section 3)

The dollar sign (or semicolon) following the END is <u>most important</u> as without it, the system could possibly read the next job on the input tape. Please ensure that this character is always the <u>last</u> in your job, or, for safety's sake, add a card with an extra $ or ; on it.

The calculations in each packet are independent, as a fresh copy of the system is read from the disk every time TEST is encountered. If an error occurs during a calculation, or during read-in, then computation of the current packet ceases, and the system proceeds to the next packet.

The <u>system name</u> on the No. 1 and No. 2 cards should be as follows:

For full gamma and index algebra (but no differentiation, determinants or greatest common divisor calculations): LISP8

For system without gamma and index algebra: LISP6

N.B. The user should read carefully the instructions on spacing (Section 2.8). Also, if the Burroughs ALGOL character set is used, a card BMODE NIL should be inserted between the relevant TEST and BEGIN cards.

The following example shows the complete card deck necessary to run one of the examples in Section 3.1 on the Stanford 7090.

```
730 ⟨No. 1 card⟩
$JOB  0000  LISP6 1      400   HEARN ⟨No. 2 card⟩
         TEST
BEGIN NIL STOP
ORDER X$ MAKE P= X**2+Y**2, Q= X**2-Y**2 $
SM DF(P,X) $ SM DF(P/Q,X,Y) $
SM DF(LOG(P)/X**2,X,Y)$
SM DF((X+2*Y)*DET((P,Q),(X+Y,Y)),X,Y) + DF(DET((X**2,Y),
   (Y**2,X)),X,Y) $
END $$$
```

4.4.2 REDUCE Job Setup for Stanford PDP-6

REDUCE is stored as a 32K system for the Stanford PDP-6 with file-name REDUCE on a DECtape available on request. To input the program, the user types

```
RUN ⟨DECtape drive number⟩ REDUCE ⟨carriage return⟩
```

which loads the system in LISP mode. The system will return two line feeds when loading is complete.

Input to a REDUCE job may come from a teletype or from a file previously setup on a DECtape. If the command

```
(BEGIN)
```

is typed on the teletype in LISP mode the system expects further input from that teletype in the form of REDUCE instructions (which will be referred to as REDUCE mode). The user should wait for the system to evaluate each instruction before typing in the next. Nearly all instructions print a single asterisk when the calculation is complete. END⟨space⟩ returns control to LISP.

If the command

```
(BEGIN ⟨DECtape number⟩ ⟨filename⟩)
```

is typed, the system evaluates REDUCE instructions on the file ⟨filename⟩, until the end of the file is reached or an END is encountered. In the former case, the system prints one asterisk on the user's teletype and expects further REDUCE input from the teletype, whereas in the latter case control returns to LISP and the teletype, and three asterisks are printed. The examples in Section 3 were run in this manner.

If an error occurs during a calculation, control returns to LISP or the time-sharing monitor, depending on the error. To return to REDUCE mode the user must type (BEGIN) in the former case, preceded by S (to return to LISP) in the latter. Replacement tables are not destroyed by errors.

A sequence of output control characters may be typed in any order after BEGIN (and before the DECtape drive number, if used). They are

L	output on line-printer
N	suppress printing of instruction name and its arguments (but not its value)

Erasing Errors on Input

a) The ⟨rubout⟩ key may be used to delete typed input as follows:

(i) If ⟨rubout⟩ is typed before the space after an instruction name, the whole instruction name (complete or incomplete) is deleted.

(ii) ⟨rubout⟩ has no effect on the instruction name once the space following it is typed. It then deletes one character from the argument input buffer each time it is typed, until the buffer is empty, after which it has no effect.

b) The character % if typed after the space following the instruction name, will delete the whole instruction, including the instruction name.

N.B. The user should read carefully the instructions on spacing (Section 2.8).

The following is a copy of the complete teletype output produced in loading REDUCE and running one of the examples in Section 3.1 on the Stanford PDP-6. In this case, the user has communicated directly with the system rather than setting up a file beforehand. The tape with the REDUCE system was on unit DTA4. Characters typed by the user are underlined.

```
.RUN DTA4 REDUCE⟨carriage return⟩

(BEGIN N)

FACIT ;
*

SM DF((X↑2+Y↑2)/(X↑2-Y↑2),X,Y) ;


(    -    8. * X**3. * Y    -    8. * X * Y**3.) / (X**6.    -    3. * X**4
. * Y**2.    +    3. * X**2. * Y**4.    -    Y**6.)



*
END ⟨space⟩
***
```

REFERENCES

1. J. McCarthy, et al. LISP 1.5 Programmer's Manual. Computation Center and Res. Lab. of Electronics, MIT, Cambridge, Mass., 1960.

2. A. C. Hearn, Communications of the A.C.M. 9, 573 (1966).

3. J. D. Bjorken and S. D. Drell, "Relativistic Quantum Mechanics" (McGraw-Hill, New York, 1965).

4. G. E. Collins, Communications of the A.C.M. 9, 578 (1966).

5. G. E. Collins, Journal of the A.C.M. 14, 128 (1967).