DECUS
PROGRAM LIBRARY

DECUS NO.	10-21b
TITLE	REDUCE 2
AUTHOR	Anthony C. Hearn
COMPANY	Computer Science Department The University of Utah Salt Lake City, Utah
DATE	December 1973
SOURCE LANGUAGE	LISP

UCP-2.2

December 1973

DECUS Program Library Write-up

DECUS NO 10-21b

REDUCE 2 IMPLEMENTATION GUIDE FOR STANFORD AI LISP 1.6 ON THE PDP-10

by

Anthony C. Hearn

University of Utah

Work supported in part by the National Science Foundation under Grant No. GJ-32181 and by the Advanced Research Projects Agency of the Office of the Department of Defense under Contract No. DAHC 15-73-C-0363.

1. INTRODUCTION

This memo describes the REDUCE DECtapes and procedures for generating a SAV file of REDUCE and testing it. REDUCE can be generated and run on any PDP-10 with the DEC operating system and at least 42K of user core.

2. DESCRIPTION OF THE REDUCE SYSTEM DECTAPES

REDUCE is distributed on two DECtapes. The first, which we shall refer to as the LISP system tape, contains the eleven ASCII files necessary for generating the Utah version of Stanford LISP 1.6. The names and contents of these files are as follows:

ALVINE.LSP	LISP part of LISP editor (ALVINE) source
ALVINE.MAC	MACRO part of LISP editor (ALVINE) source
COMPLR	LISP source for LISP compiler
GRIN	LISP source for LISP pretty printing functions
LAP	LISP definitions of the LISP assembly program (LAP)
LISP.MAC	MACRO source for LISP interpreter
LISP.LSP	LISP auxiliary functions used with interpreter
LOADER.MAC	MACRO source for LISP loader
SMILE	LISP definitions of some useful LISP functions
SYMMAK.MAC	MACRO source for LISP symbol table generation
TRACE	LISP definitions of some debugging functions

The second tape which we shall refer to as the REDUCE system tape contains the nine ASCII files necessary for the generation and testing of REDUCE itself. The names and contents of these files are as follows:

ARITH.MAC	MACRO part of the REDUCE arithmetic package
ARITH.LSP	LISP part of the REDUCE arithmetic package
BOOT.LSP	LISP functions required for bootstrapping REDUCE
GUIDE	this Guide
REDH.LSP	LISP definitions of the REDUCE debugging package
REDT.RED	tests of the REDUCE system written in REDUCE
REDUCE.RED	source of REDUCE written in REDUCE
SCAN.MAC	MACRO definition of the REDUCE input scanner
UPDATE.MAC	MACRO definition of the update program

The following documentation should be provided with the DECtapes:

1) REDUCE 2 Implementation Guide for Stanford AI LISP 1.6 on the PDP-10 (UCP 2) (i.e., this Guide)
2) Stanford AI LISP 1.6 Manual (SAILON 28.7).
3) REDUCE 2 User's Manual, Second Edition (UCP 19)

3. ASSEMBLY OF REDUCE

The generation of a REDUCE dump file proceeds in three stages. These stages, which are described in subsequent sub-sections, are as follows:

1) Assembly and Execution of the update program

2) Generation of the LISP interpreter and compiler and associated files

3) Generation and testing of the REDUCE SAV file

We assume that a disk is available for temporary storage of files. If this is not true, the author should be contacted for instructions on using DECtapes instead.

As we have prepared this guide for printing on a teletype, we are unable to follow the usual convention of underlining the system output and prompts. However, we hope that anyone used to the DEC operating system will not be too confused by this omission. As a general rule, the system prompt characters (. and *) are displayed in column 1 in the prototypical dialogs below and the user input begins in column 2. Computer responses where given are enclosed in angle brackets. In other words, the user should NOT type the . and * characters in column 1 in the procedures described below or type anything enclosed in angle brackets.

3.1 Assembly and Execution of the Update Program

The file UPDATE.MAC on the REDUCE system tape is a MACRO program which is used to update source files according to selected control statements. The updated file is input compatible with the update program.

3.1.1 Format of Update Control Statements

Changes are made in the original file according to update control statements in a file with the same name as the original file but with an extension UPD. The format of these statements is as follows:

./ D m n

D (delete) specifies that records are to be deleted from the original file.

m is the sequence number of the first record to be deleted.

n is the sequence number of the last record to be deleted. If omitted, only one record is deleted.

./ I m

I (insert) specifies that the records immediately following this statement until the next control statement are to be inserted in the original file.

m specifies the sequence number of the record after which the new records are to be inserted.

./ N m

N (number) is used to set a sequencing increment for records inserted in the new file by an I or R command. The actual increment is specified by m, which is initially set to 0.

./ R m n

R (replace) specifies that records following this statement are to be inserted in the original file in place of the records specified.

m is the sequence number of the first record to be replaced.

n is the sequence number of the last record to be replaced. If omitted, only one record is replaced.

In the above update control statements, the control characters . and / must always be in the first and second column respectively. In particular, the . is NOT a system prompt and must be provided by the user. The remaining characters and numbers may be used free field.

3.1.2 Assembly and Loading of the Update Program

This is accomplished by the following steps.

```
.R MACRO<cr>
*UPDATE.REL←UPDATE.MAC<cr>
<appropriate messages>
*↑C
.R LOADER<cr>
*UPDATE<altmode>
<appropriate messages>
↑C
EXIT

.SAVE DSK: UPDATE<cr>
```

UPDATE.SAV should now be copied to SYS:

3.1.3 Execution of the Update Program

The UPDATE program may now be used to update any file on the system tapes (except for the large MACRO files on the LISP system tape which have no line numbers) or any file which has been previously updated by this program. The program asks for the name of the original file and the updated file. It assumes as mentioned above that the update control statements are in a file with the same name as the original file and an extension UPD. For example, to update the file REDUCE.RED, the update control statements would be in a file REDUCE.UPD. If we wanted to give the updated file the name RED, the following steps would be followed:

```
.R UPDATE<cr>
<Enter INP  file:> REDUCE.RED<cr>
<Enter OUTP file:> RED<cr>
```

3.2 Generation of the LISP Interpreter and Compiler

The generation of the LISP system proceeds in 9 steps. The instructions contained here are a modification of those found in Appendix P of the Stanford AI LISP 1.6 Manual.

To begin with the files on the LISP system tape should be copied to DSK:. The generation of the LISP system is now accomplished by the following prototypical dialogs:

3.2.1 Assembly of MACRO Files

```
.R MACRO<cr>
*LISP.REL←LISP.MAC<cr>
<appropriate messages>
*LOADER.REL←LOADER.MAC<cr>
<appropriate messages>
*SYMMAK.REL←SYMMAK.MAC<cr>
<appropriate messages>
*ALVINE.REL←ALVINE.MAC<cr>
<appropriate messages>
*
```

These assemblies take a total CPU time of about 3 1/2 minutes.

3.2.2 Generation of the LISP Interpreter

```
.R LOADER<cr>
*LISP<altmode>
<appropriate messages>
↑C

.SAVE DSK:LISP 10<cr>
```

3.2.3 Generation of the LISP Loader

```
.R LOADER<cr>
*LOADER<altmode>
<appropriate messages>
↑C

.START<cr>

<This creates the mode 17 file DSK:LISP.LOD>
```

3.2.4 Generation of the LISP Loader Symbol Table

```
.R LOADER<cr>
*DSK:LISP/J/D,DSK:SYMMAK<altmode>
<appropriate messages>
```

↑C

.START<cr>

<This creates the file DSK:LISP.SYM>

3.2.5 Saving Files on SYS:

At this point, the following files should be copied to SYS:

LISP.SAV, LISP.LSP, LISP.LOD, GRIN, LAP, SMILE and TRACE

The files GRIN, SMILE and TRACE whose use is described in the Stanford AI LISP 1.6 Manual are not actually required by REDUCE and may therefore be deleted if they are not needed for other purposes.

If it is not possible to use SYS:, then the following assignment should be made at this point:

.ASSIGN DSK: SYS:<cr>

3.2.6 Generating ALVINE (The LISP Editor)

.R LISP<cr>

ALLOC? <cr>

<Stanford Lisp 1.6 (Utah version) 1-May-73>
*(DSKIN (ALVINE.LSP))<cr>

*(LOAD)<cr>
*ALVINE<altmode>

<LOADER 2K CORE>

*(GETSYM SUBR ALVINE)<cr>

<message>
*(ALVINE)<cr>

<NIL>

<This creates the file DSK: LISP.ED>

Copy the file LISP.ED to SYS:

3.2.7 Generation of the LISP Compiler

3.2.7.1 If the previous version of the compiler exists, do

```
.R COMPLR 35<cr>
```

and go to step 3.2.8.

3.2.7.2 If there is not an older version of the compiler, do

```
.R LISP 34<cr>

ALLOC? <cr>

<Stanford Lisp 1.6 (Utah version) 1-May-73>
*(DSKIN COMPLR)<cr>

<this loads everything>
```

and continue with step 3.2.8.

3.2.8 Actual Compilation of the LISP Compiler

```
*(COMPL COMPLR)<cr>

<appropriate messages>

<This generates the file DSK: COMPLR.LAP>
```

3.2.9 Loading the Compiled LISP Compiler

```
.R LISP 30<cr>

ALLOC? Y
FULL. WDS=3000<space>
BIN.PROG.SP=14400<space>
SPEC.PDL=<space>
REG. PDL=<space>
HASH=475<space>

<Stanford Lisp 1.6 (Utah version) 1-May-73>
*(DSKIN (COMPLR.LAP))<cr>
<various messages>
*(CINIT)<cr>

↑C

.SAVE DSK: COMPLR<cr>
```

Copy the file COMPLR.SAV to SYS:

Users should note that it takes about 1 hour CPU time to build the compiler from the system tape source files.

At this point, all files which have not been copied to SYS: may be deleted.

3.3 Generation of the REDUCE SAV File

The generation of REDUCE proceeds in four main steps. As in the last Section, we begin by copying all REDUCE system tape files to DSK:. The generation of the REDUCE system is now accomplished by the following prototypical dialogs:

3.3.1 Assembly of MACRO Files

```
.R MACRO<cr>
*ARITH.REL←ARITH.MAC<cr>
<appropriate messages>
*SCAN.REL←SCAN.MAC<cr>
<appropriate messages>
*
```

3.3.2 Compiling REDH.LSP

```
.R COMPLR<cr>

<LISP COMPILER>

*(COMPL (REDH . LSP))<cr>
<appropriate messages>
*
```

3.3.3 Building the REDUCE Symbolic Mode Program (RLISP)

In order to translate the REDUCE source into LISP, it is convenient to use a program which contains only the symbolic (non-algebraic) parts of REDUCE. This particular program, which we shall refer to as RLISP to avoid confusion with the full REDUCE program, also provides the user with a useful higher level language for the development of symbolic programs. The generation of this program is accomplished by the following steps:

1) Create a file RLISP by updating the file REDUCE.RED using the following update file:

```
./ D 25000 89000
```

If the previous version of RLISP.SAV exists on SYS:, then go to Step 4.

2) Create a file RED by updating the file REDUCE.RED using the following update file:

```
./ D 1 10610
./ D 19690 22490
./ D 23440 89000
```

3) Run the following job:

```
.R LISP 35<cr>
ALLOC? <cr>

<Stanford Lisp 1.6 (Utah version) 1-May-73>

*(DSKIN (BOOT . LSP))<cr>
<appropriate messages>
*(LOAD T)<cr>
*DSK:SCAN<altmode>

<LOADER 1K CORE>

*(INITL)<cr>
*DSKIN 'RED;<cr>

<NIL>

*BEGIN2();<cr>
*ON DEFN;<cr>
*DSKOUT 'REDLSP;<cr>
*DSKIN 'RLISP;<cr>
<appropriate messages>
*
```

This step takes approximately 5 minutes CPU time.

Now go to Step 5.

4) Translate the REDUCE source in the file RLISP to LISP by the following procedure:

```
.R RLISP<cr>
<REDUCE 2 (<system date>) ...>

*ON DEFN;<cr>

*OUT REDLSP;<cr>
*IN RLISP;<cr>
<various messages>
*END;<cr>
<ENTERING LISP...>
```

5) Compile the REDLSP file by the following procedure:

```
.R COMPLR<cr>

<LISP COMPILER>

*(SETQ IBASE (PLUS 7 3))<cr>

<12>
*(COMPL REDLSP)<cr>

<various messages>
```

This step takes approximately 2 1/2 minutes CPU time.

6) Generate the RLISP SAV file by the following procedure:

```
.R LISP 22<cr>

ALLOC? Y
FULL WDS=1300<space>
BIN.PROG.SP=13000<space>
SPEC.PDL=700<space>
REG. PDL=700<space>
HASH=475<space>

<Stanford Lisp 1.6 (Utah version) 1-May-73>

*(OUTC (OUTPUT DSK: (LST . TMP)))<cr>
*(SYSIN LAP) (DSKIN (REDLSP . LAP) (REDH . LAP))<cr>
*(LOAD T)<cr>
*DSK:SCAN<altmode>

<LOADER 1K CORE>

*(INITL)<cr>

<NIL>
*(SETQ IMODE* @SYMBOLIC)<cr>

<SYMBOLIC>
*↑C
.SAVE DSK: RLISP<cr>
```

Copy the file RLISP.SAV to SYS:.

This step takes approximately 1 1/2 minutes CPU time.

3.3.4 Building the REDUCE SAV file

This is accomplished by the following 4 steps:

1) Update the file REDUCE.RED (if necessary) to produce a file REDUCE. If no updating is required, simply copy REDUCE.RED to REDUCE.

2) Translate the REDUCE source into LISP by the following procedure:

```
.R RLISP 30<cr>
<REDUCE 2 (<system date>) ...>

*ON DEFN;

*OUT REDLSP;<cr>
*IN REDUCE;<cr>
<various messages>
*END;<cr>
<ENTERING LISP...>
```

This step takes approximately 6 1/2 minutes CPU time.

3) Compile the REDLSP file by the following procedure:

```
.R COMPLR 35<cr>

<LISP COMPILER>
*(SETQ IBASE (PLUS 7 3))<cr>

<12>
*(COMPL REDLSP)<cr>

<various messages>
```

This step takes approximately 10 minutes CPU time.

4) Generate the REDUCE SAV file by the following procedure:

```
.R LISP 42<cr>

ALLOC? Y
FULL WDS=4000<space>
BIN.PROG.SP=47000<space>
SPEC.PDL=1000<space>
REG. PDL=1000<space>
HASH=475<space>

<Stanford Lisp 1.6 (Utah version) 1-May-73>

*(OUTC (OUTPUT DSK: (LST . TMP)))<cr>
*(SYSIN LAP) (DSKIN (ARITH . LSP) (REDLSP . LAP) (REDH . LAP))<cr>
*(LOAD T)<cr>
*DSK:SCAN,DSK:ARITH<altmode>
```

```
<LOADER 2K CORE>

*(INITL)<cr>

<NIL>
*↑C
.SAVE DSK: REDUCE<cr>
<Job saved>
↑C
.
```

Copy the file REDUCE.SAV to SYS:.

This step takes approximately 4 minutes CPU time.

Users should note that this version of REDUCE is saved with a very small workspace. It will therefore often be necessary to specify more core for your job by following the instructions given in Appendix B.1 of the REDUCE 2 User's Manual.

3.3.5 Assembly of REDUCE without High Energy Physics Package

The version of REDUCE generated in the previous section contains the functions and operators necessary for solving problems in high energy physics. If you do not need these functions, you can assemble the system in a minimum 40K core partition. The assembly of such a system is accomplished in the following steps:

1) Create a file REDUCE by updating REDUCE.RED with the following update file:

```
./ D 65000 71560
```

2) Repeat steps 2) and 3) in previous Section (3.3.4)

3) Generate the REDUCE SAV file by the following procedure:

```
.R LISP 40<cr>

ALLOC? Y
FULL WDS=4000<space>
BIN.PROG.SP=42000<space>
SPEC.PDL=1000<space>
REG. PDL=1000<space>
HASH=475<space>

<Stanford Lisp 1.6 (Utah version) 1-May-73>
```

```
*(OUTC (OUTPUT DSK: (LST . TMP)))<cr>
*(SYSIN LAP) (DSKIN (ARITH . LSP) (REDLSP . LAP) (REDH . LAP))<cr>
*(LOAD T)<cr>
*DSK:SCAN,DSK:ARITH<altmode>

<LOADER 2K CORE>

*(INITL)<cr>

<NIL>
*↑C
.SAVE DSK: REDUCE<cr>
<Job saved>
↑C
.
```

Copy the file REDUCE.SAV to SYS:.

If RLISP.SAV already exists, this version of REDUCE takes about 15 minutes total CPU time to build.

3.3.6 Deleting unnecessary Files

After the SAV files have been copied to SYS:, all files left on DSK: may be deleted.

4. TESTING THE REDUCE SYSTEM

A series of tests of the assembled REDUCE system are available on the file REDT.RED on the REDUCE system tape. To use these tests, the following job should be run:

```
.R REDUCE 50<cr>

REDUCE 2 (<current system date>) ...

*IN REDT.RED;<cr>
```

The output from these tests should be self- explanatory. The various options on directing output from REDUCE jobs to other devices, as explained in Appendix B.1 of the REDUCE 2 User's Manual, may be followed if desired. The tests have a total run time of about 3 minutes.

5. PROGRAM CHANGES

Any person receiving the REDUCE 2 program should send his (or her) name and installation address to

REDUCE Secretary
Department of Physics
University of Utah
Salt Lake City, Utah 84112 USA

Telephone (801) 581-8502

It is most important that this be done so that we can advise you direct of any changes which are made to the system.

6. INQUIRIES AND REPORTING OF ERRORS

Any inquiries regarding the assembly or operation of REDUCE should also be directed to the above address. Suspected errors should be accompanied by the relevant job output and a copy of the input program, preferably on punched paper tape.

REDUCE 2

USER'S MANUAL*

by

Anthony C. Hearn
University of Utah
Salt Lake City, Utah 84112 USA

Second Edition

*Work supported in part by the National Science Foundation under Grant No. GJ-32181 and by the Advanced Research Projects Agency of the Office of the Department of Defense under Contract No. DAHC 15-73-C-0363.

ABSTRACT

This manual provides the user with a description of the algebraic programming system REDUCE 2. The capabilities of this system include:

1) expansion and ordering of polynomials and rational functions,
2) symbolic differentiation,
3) substitutions and pattern matching in a wide variety of forms,
4) calculation of the greatest common divisor of two polynomials,
5) automatic and user controlled simplification of expressions,
6) calculations with symbolic matrices,
7) a complete language for symbolic calculations, in which the REDUCE program itself is written,
8) calculations of interest to high energy physicists including spin 1/2 and spin 1 algebra,
9) tensor operations.

UPDATES TO MANUAL

This copy of the REDUCE 2 User's Manual includes all updates through December 15, 1973.

TABLE OF CONTENTS

1. INTRODUCTION

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

1) expansion and ordering of polynomials and rational functions,
2) symbolic differentiation,
3) substitutions and pattern matching in a wide variety of forms,
4) calculation of the greatest common divisor of two polynomials,
5) automatic and user controlled simplification of expressions,
6) calculations with symbolic matrices,
7) a complete language for symbolic calculations, in which the REDUCE program itself is written,
8) calculations of interest to high energy physicists including spin 1/2 and spin 1 algebra,
9) tensor operations.

There are several levels at which REDUCE may be used and understood. The beginning user can acquire an operating knowledge of the system by studying Section 2 of this manual, which contains details of the basic structure of programs. Having mastered this, he should then study Section 3, which describes the facilities available for manipulating algebraic expressions. A more advanced user must understand Sections 4 and 5, which describe the matrix handling routines and some advanced commands. Finally, to become a complete expert, the user will have to learn LISP 1.5 and study the material on the symbolic mode of operation in Section 6.

Additional material of interest may be found in the Appendices. Appendix A gives the user a useful summary of the system commands and other properties of the system. Appendix B gives instructions for using REDUCE at various computer installations. This information may of course need modification for your computer. Finally, Appendix C contains details of the high energy physics commands for those interested.

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

2. STRUCTURE OF PROGRAMS

2.1 Preliminary

A REDUCE program consists of a set of functional commands which are evaluated sequentially by the computer. These commands are built up from declarations, statements and expressions. Such entities are composed of sequences of numbers, variables, operators, strings, reserved words and delimiters (such as commas and parentheses), which in turn are sequences of basic characters.

2.2 The REDUCE Standard Character Set

The basic characters which are used to build up REDUCE symbols are the following:

i) The 26 upper case letters A through Z
ii) The 10 decimal digits 0 through 9
iii) The special characters ! " $ % ' ( ) * + , - . / : ; < > =

Programs composed from this standard set of characters will run in any available REDUCE system. In addition, several implementations of REDUCE (for example, on the PDP-10) use additional characters to represent some of the operators in the system. The local operating instructions for the given computer such as those in Section B of this manual should be consulted for the meaning of these special characters. However, for generality in exposition we shall limit ourselves to the standard character set in the body of this manual.

2.3 Numbers

Numbers in REDUCE may be of two types; integer and real. Integers consist of a signed or unsigned sequence of decimal digits written without a decimal point.

e. g. -2, 5396, +32

There is no practical limit on the number of digits permitted as arbitrary precision arithmetic is used.

Real numbers may be written in two ways;

i) as a signed or unsigned sequence of 1-9 decimal digits with an embedded 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 0.32E2 and 320.E-1 are all representations of 32.

Restriction:

The unsigned part of any number may NOT begin with a decimal point.

i. e. NOT ALLOWED .5 -.23 +.12

2.4 Identifiers

Identifiers in REDUCE consist of one to twenty-four alphanumeric characters (i.e. upper case alphabetic letters or decimal digits) the first of which must be alphabetic.

e. g. A AZ P1 Q23P AVERYLONGVARIABLE

Identifiers are used as variables, labels and to name arrays, operators and procedures.

Restrictions:

Reserved words in REDUCE (see Section 2.10) may not be used as identifiers. No spaces may appear within an identifier, and an identifier may not extend over a line of text.

2.5 Variables

Variables are a particular type of identifier, and are specified by name and type. Their names must be allowed identifiers. There are several variable types allowed, and these are discussed later, beginning in Section 2.12.1.

2.5.1 Reserved Variables

Several variables in REDUCE have a particular value which cannot easily be changed by the user. These variables are as follows:

I	square root of -1. All powers of I are automatically replaced by the appropriate combination of -1 and I.
E	base of natural logarithms

2.6 Operators

Operators in REDUCE 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

X = Y AND W MEMBER Z

The following infix operators are built into the system.

<infix operator>::= :=|OR|AND|NOT|MEMBER|=|NEQ|EQ|>=|>|<=|<|+|-|*|/|**|.

The use of the EQ operator is explained in Section 6 and the . operator in Section 6 and Appendix C.

These operators may be further divided into the following sub classes

```
<assignment operator> ::= :=
<logical operator>    ::= OR|AND|NOT|MEMBER
<relational operator> ::= =|EQ|NEQ|>=|>|<=|<
<arithmetic operator> ::= +|-|*|/|**
<symbolic operator>   ::= .
```

For compatibility with the intermediate language used by REDUCE, each special character infix operator has an additional alphanumeric identifier associated with it. These identifiers may be used interchangably with the corresponding infix character(s) on input. This correspondence is as follows:

:=	SETQ
=	EQUAL
>=	GEQ
>	GREATERP
<=	LEQ
<	LESSP
+	PLUS
-	DIFFERENCE (unary MINUS)
*	TIMES
/	QUOTIENT (unary RECIP)
**	EXPT
.	CONS

The above operators are assumed to be binary, except NOT which is unary and + and * which are nary. In addition, - and / may be used in a unary position. Any other operator is parsed as a binary operator using a left procedure grouping. Thus A/B/C is interpreted as (A/B)/C. There are two exceptions to this latter rule, namely := and . which have a right precedence grouping. Thus A:=B:=C is interpreted as A:=(B:=C).

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

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

e. g. COS(U)
DF(X**2,X)

Parentheses may be omitted if the operator is unary.

e. g. COS Y and COS(Y) are equivalent.

Such unary prefix operators have a precedence higher than any infix operator.

Infix operators may also be used in a prefix format on input. On output, however, they will always be printed in infix form.

Some prefix operators are also built into the system. These are as follows:

2.6.1 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. The remaining arguments specify the differentiation variables and the number of times they are applied by the following syntax:

DF(<expression>,<variable>,<number>,...,<variable>,<number>)

The <number> may be omitted if it is 1.

e. g. DF(Y,X) = dY/dX

$$DF(Y,X,2) = d^2Y/dX^2$$

$$DF(Y,X1,2,X2,X3,2) = d^5Y/dX_1^2\,dX_2\,dX_3^2$$

2.6.2 COS, LOG, SIN

These elementary functions are included in the system with the following properties:

COS (-X) = COS (X)
SIN (-X) = - SIN (X)
COS (0) = 1
SIN (0) = 0
LOG (E) = 1
LOG (1) = 0

Their derivatives are also known to the system. The user can also add further rules for the reduction of expressions involving these operators by the methods described in Sections 2.15 and 5.2.

2.6.3 SUB

This operator is described in Section 2.15.

The user may add new prefix operators to the system by the declaration OPERATOR.

e. g. OPERATOR H,G1,ARCTAN;

adds the prefix operators H, G1 and ARCTAN to the system.

2.7 Strings

Strings are used only in output statements. A string consists of any number of characters enclosed in double quotes.

e. g. "A STRING".

2.8 Comments

Comments are useful for including explanatory material at various points in a program. They may be used in the following form:

COMMENT <any sequence of characters not including a terminator>
<terminator>

where

<terminator> ::= ;|$

e. g. COMMENT THIS IS A COMMENT;

Such comments are equivalent on input to an space. In addition, the sequence of symbols

END <any sequence of symbols not including a terminator or the reserved words END, or ELSE or UNTIL>

is equivalent to the reserved word END.

2.9 Expressions

REDUCE expressions may be of several types and consist of syntactically allowed sequences of numbers, variables, operators, left and right parentheses and commas. The most common types are as follows:

2.9.1 Numerical Expressions

These consist of syntactically allowed combinations of integer or real variables, arithmetic operators and parentheses. They evaluate to numbers.

Examples:

```
2
I + J - 2 * I**2
```

are numerical expressions if I and J are integers.

2.9.2 Scalar Expressions

These consist of scalar variables and arithmetic operators and follow the normal rules of scalar algebra.

Examples:

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

2.9.3 Boolean Expressions

These are expressions which return a truth value. In REDUCE, the reserved word NIL represent the truth value 'false'. Any other expression represents 'true'. So in a sense any expression is a boolean expression because all expressions return a value. However, a proper boolean expression has the syntactical form:

<expression> <relational operator> <expression>

or

<boolean expression> <logical operator> <boolean expression>

They are used mainly in the IF and FOR statements described in Sections 2.11.2 and 2.11.3.

Examples:

```
J<1
X>0 or  X=-2
```

2.9.4 Equations

In the remainder of this manual, we shall refer to the expression

<expression> = <expression>

as an equation.

2.10 Reserved Words

Certain words are reserved in REDUCE. They may only be used in the manner described in this manual. These words are as follows:

<reserved word> ::= BEGIN|DO|ELSE|END|FOR|FUNCTION|GO|GOTO|IF|LAMBDA|
NIL|PRODUCT|RETURN|STEP|SUM|TO|UNTIL|WHILE|

2.11 Statements

A statement is any allowed combination of reserved words and expressions, and has the syntax

<statement> ::= <expression>|<proper statement>

The following sub-sections describe some proper statements in REDUCE.

2.11.1 Assignment Statements

These statements have the following syntax

<assignment statement> ::= <expression> := <statement>

e. g. A1:=B+C
H(X,Y):=X-2*Y

By analogy with numerical assignments in ALGOL, an assignment statement sets the left hand side of the statement to the algebraic value of the right hand side. Unfortunately, the algebraic evaluation of an expression is not as unambiguous as the corresponding numerical evaluation. This algebraic evaluation process is generally referred to as 'simplification' in the sense that the evaluation usually but not always produces a simplified form for the expression. In REDUCE, the default evaluation of an expression involves expansion of the expression and collection of like terms, ordering of the terms, evaluation of derivatives and other functions and substitution for any expressions which have values assigned or declared (Sections 2.15 and 5.2). In many cases, this is all that the user needs.

However, the user can exercise some control over the way in which the evaluation is performed by various declarations available to him. These declarations are explained in Section 3.3.

If a real (floating point) number is encountered during evaluation, the system will normally convert it into a ratio of two integers, and print a message informing the user of the conversion. If the user wants to use real arithmetic, he can inhibit this conversion by the command ON FLOAT;. This use of the ON declaration is explained in Section 2.12.3.

The results of the evaluation of an expression are printed if a semicolon is used as a delimiter. Because it is not usually possible to know in advance how large an expression will be, no explicit format statements are offered to the user. However, a variety of output declarations are available so that the output can be produced in a variety of forms. These output declarations are explained in Section 3.4.

It is also possible to write an assignment in the form

<expression> := <expression> := ... := <expression> := <statement>

In this form, each <expression> is set to the value of the <statement>.

2.11.2 Conditional Statements

The conditional statement has the following syntax:

<conditional statement> ::= IF <boolean expression> THEN <statement>
ELSE <statement>

Its use is obvious. The ELSE clause is optional.

2.11.3 FOR Statements

The FOR statement is used to define a variety of program loops. Its general syntax is as follows:

```
FOR <variable>:=<arithmetic expression> STEP <arithmetic expression>
                                        (DO <statement>
   (UNTIL <arithmetic expression>)      (
   (                              )     (SUM <algebraic expression>
   (WHILE <boolean expression>    )     (
                                        (PRODUCT <algebraic expression>
```

The DO version of the FOR statement is the normal ALGOL usage, and is similar to the FORTRAN DO statement. Its value is 0. The SUM and PRODUCT versions respectively form the sum and product of the relevant algebraic expression over the defined range. They return the value of the computed sum or product.

The <variable> within the FOR statement is assumed INTEGER. Its value during the calculation of the statement is independent of its value outside, so that I can be used in this context, even though I normally stands for the square root of -1.

Examples:

We assume that the declaration ARRAY A(10); has been made in the following examples. This declaration is explained in Section 2.12.2.

(i) To set each element A(I) of the array to X**I we could write

```
FOR I:=0 STEP 1 UNTIL 10 DO A(I):=X**I$
```

As a further convenience, the common construction

```
STEP 1 UNTIL
```

may be abbreviated by a colon. Thus we could write instead of the above

```
FOR I:=0:10 DO A(I):=X**I$
```

Since the assignments in this calculation are not made at the top level, they are not printed. If the user desires this, he can use the WRITE statement for this purpose as in

```
FOR I:=0:10 DO WRITE A(I):=X**I;
```

Further uses of WRITE are described in Section 3.2.4.

(ii) To sum the squares of the even positive integers through 50, we could write

```
FOR I:=2 STEP 2 UNTIL 50 SUM I**2;
```

(iii) To set X to the factorial of 10 we could write

```
X := FOR I:=1:10 PRODUCT I;
```

Alternatively, we could set the element A(I) to I! by the statements

```
A(0):= 1$ FOR I:=1:10 DO A(I):=I*A(I-1)$
```

2.11.4 GO TO Statements

GO TO (or GOTO) statements are used to perform an unconditional transfer to a label in a compound statement (Section 2.11.5). They have the syntax:

```
<go to statement> ::= GO TO <label>
<label> ::= <variable>
```

Restrictions:

GO TO statements may only occur within a compound statement. They may NOT occur at the top level of your program. Furthermore, they can only reference labels within the local block in which they are defined.

2.11.5 Compound Statements

A compound statement is defined by the following syntax

```
<compound statement> ::= BEGIN <compound tail>
<compound tail> ::= <unlabeled compound tail>
                  |<label>:<compound tail>
<unlabeled compound tail> ::= <statement> END
                  | <statement> <terminator> <compound tail>
<label> ::= <identifier>
<terminator> ::= ;|$
```

```
e. g. X :=  BEGIN INTEGER M;
                M:=1$
            L1: IF N=0 THEN RETURN M;
                M:=M*N$
                N:=N-1$
                GO TO L1
            END OF BLOCK;
```

will assign the factorial of a preassigned INTEGER N to X.

2.11.6 RETURN Statements

The RETURN statement allows for a transfer out of a compound statement to the next highest program level. It may be used alone, in which case the statement returns 0.

```
e. g.,  RETURN (X+Y);
        RETURN M;
        RETURN;
```

Restriction:

RETURN statements may only occur within a compound statement.They may NOT occur at the top level of your program.

2.12 Declarations

Declarations are a particular type of statement used to set flags, make type declarations and define procedures. PROCEDURE declarations are discussed in Section 2.18. Some other REDUCE declarations are described in the following sub-sections.

2.12.1 Variable Type Declarations

These declarations tell the system how various identifiers are to be processed. Types allowed include INTEGER, REAL and SCALAR.

```
e. g.   INTEGER M,N;
        REAL M1;
        SCALAR X,Y;
```

Type declarations may be made at any level in a program, and apply only to the particular program block in which they occur. Variables not declared are assumed SCALAR. This is the basic symbolic variable type. All such variables are given the initial value of 0.

2.12.2 Array declarations

Arrays in REDUCE are defined similar to FORTRAN dimension statements.

```
e. g. ARRAY A(10),B(2,3,4);
```

Their indices each range from 0 to the value declared. An element of an array is referred to in standard FORTRAN notation,

```
e. g. A(2)
```

All array elements are initialized to 0 at declaration time.

Array elements may appear in the left side of assignment statements as in the examples in Section 2.11.3.

2.12.3 Mode Handling Declarations

Two declarations are offered to the user for turning on or off a variety of flags in the system. The declarations ON and OFF take a list of flag names as argument and turn them on and off respectively.

```
e. g.   ON FLOAT,GCD;
        OFF LIST;
```

The use of these flags and others available to the user is explained later in this manual.

2.13 Commands

A command is an order to the system to do something. It has the syntax

```
<command> ::= <statement><terminator>|<proper command>
```

<proper command> ::= <command name><space>
<statement>,...,<statement><terminator>

A variety of commands are discussed in the sections which follow.

2.14 File Handling Commands

In many applications, it is desirable to load previously prepared REDUCE files into the system, or to write output on varying devices. REDUCE offers three commands for this purpose, namely, IN, OUT, and SHUT.

2.14.1 IN

This command takes a list of file names as argument and directs the system to input each file (which should contain REDUCE commands) into the system. A file name will have a varying syntax from implementation to implementation, but in many cases will be an identifier.

e. g. IN F1,GGG; will load the files F1 and GGG.

When input comes from an external file, statements are echoed on the output device as they are read. If this facility is not required, the echoing can be prevented by turning off the flag ECHO in the relevant file.

2.14.2 OUT

This command takes a single file name as argument, and directs output to that file from then on. If the file has previously been used for output during the current job, the output is appended to the end of the file. An existing file is erased before its first use for output in a job.

To output on the terminal without closing the output file, the reserved file name T (for terminal) may be used.

e. g. OUT OFILE; will direct output to the file OFILE and
OUT T; will direct output the user's terminal.

2.14.3 SHUT

This command is used to close an output file at completion. Most systems require this action by the user, otherwise output may be lost. If a file is shut and a further OUT command issued for the same file, the file is erased before the new output is written.

2.15 Substitution Commands

An important class of commands in REDUCE is that class which defines substitutions for variables and expressions to be made during the evaluation of expressions. Such substitutions may be declared globally by the command LET and locally by use of the operator SUB.

LET is used in the form

LET <substitution list>;

where <substitution list> is a list of equations of the form:

<variable> = <expression>

or <prefix operator> (<argument>,...,<argument>) = <expression>

or <argument> <infix operator>,..., <argument> = <expression>

e. g. LET X = Y**2 + 2,
H(X,Y) = X - Y,
COS(60) = 1/2,
Y**3 = 2*Z - 3;

These substitutions will now be made for all such variables and expressions appearing in evaluations. Any operators occuring in such equations will be automatically declared OPERATOR by the system.

In each of these examples, substitutions are only made for the explicit expressions given; i.e., none of the variables may be considered arbitrary in any sense.

For example, the command

LET H(X,Y) = X - Y;

will replace H(X,Y) by X - Y, but not H(X,Z) or any other function of H. If a substitution for all possible values of a given argument of an operator is required, the declaration FOR ALL (or FORALL) may be used. The syntax of such a command is

FOR ALL <variable>,...,<variable> <LET command><terminator>

e. g. FOR ALL X,Y LET H(X,Y) = X-Y;
FOR ALL X LET K(X,Y) = X**Y;

In applying any substitutions set up by LET, the system searches the substitution expression itself for any expressions which may themselves have substitutions declared for them. Thus LET sets up an equivalence between the left hand side of the substitution and the right hand side rather than an assignment as made by an assignment statement. In other words, a substitution such as

```
LET X = X + 1;
```

is not allowed, since in substituting for X, the system will find that the variable X in the substitution expression itself has a substitution and a push-down-list overflow error will ultimately result. Similarly, a pair of substitutions

```
LET L = M + N, N = L + R;
```

are not allowed.

On the other hand, if the user wishes simply to replace every occurrence of a variable by any expression without checking that expression again for further substitutions, the operator SUB may be used. Its general form is

SUB (<substitution list>,<expression>)

as in

SUB(X=X+1,Y=1,X**2+Y**2)

This substitution is made by first simplifying the <expression>, then replacing any variables occurring on the substitution list, and finally resimplifying the result. Thus, in the above example, the result would be

X**2+2*X+2

2.16 Removing Assignments and Substitution Rules from Expressions

The user may remove all assignments and substitution rules from any expression by the command CLEAR, in the form

CLEAR <expression>,...,<expression><terminator>

e. g. CLEAR X, H(X,Y);

A whole array, such as A in Section 2.12.2, can be cleared by the command CLEAR A; An individual element of A can also be cleared by a command such as CLEAR A(3);

2.17 Adding Rules for Differentiation of User-defined Operators

An extension of the syntax of LET arguments allows for the introduction of rules for differentiation of user-defined operators. Its general form is

```
FOR ALL <var1>,...,<varn>
    LET DF(<operator><varlist>,<vari>)=<expression>
```

where <varlist> ::= (<var1>,...,<varn>)

and <var1>,...,<vari>,...,<varn> are the dummy variable arguments of <operator>.

An analogous form applies to infix operators.

We illustrate this with some examples:

```
FOR ALL X LET DF(TAN X,X)= SEC(X)**2;
FOR ALL X,Y LET DF(F(X,Y),X)=2*F(X,Y),
                DF(F(X,Y),Y)=X*F(X,Y);
```

We notice that all dummy arguments of the relevant operator must be declared arbitrary by the FOR ALL command, and that rules may be supplied for operators with any number of arguments. If no differentiation rule appears for any argument in an operator, the differentiation routines will return an expression in terms of DF as the result. For example, if the rule for the differentiation of the second argument of F is not supplied, the evaluation of DF(F(X,Z),Z) would leave this expression unchanged.

2.18 Procedures

It is often useful to name a statement for repeated use in calculations with varying parameters, or to define a complete evaluation procedure for an operator. REDUCE offers a procedural declaration for this purpose. Its general syntax is:

```
<procedural type> PROCEDURE <name><varlist>;<statement>;
```

and

```
<varlist> ::= (<variable>,...,<variable>)
```

The types permitted in REDUCE are REAL, INTEGER and ALGEBRAIC. The default type is ALGEBRAIC. All these procedures are automatically declared to be operators on definition.

In the present system, no distinction is made in the handling of these three types, although this may not be true in later versions.

Examples:

(1) The example in Section 2.11.5 could be made into an integer procedure FAC by the declaration:

```
INTEGER PROCEDURE FAC (N);
        BEGIN INTEGER M;
              M:=1$
          L1: IF N=0 THEN RETURN M;
              M:=M*N$
              N:=N-1$
```

```
                GO TO L1
        END;
```

If we now evaluate FAC (3) we get the result 6.

(2) As an example of an algebraic procedure, we define an operator LEG of two arguments which evaluates to the Legendre polynomial. We define this operator as a procedure from the generating function formula

$$p_n(x) = \frac{1}{n!}\,\frac{d^n}{dy^n}\left(\frac{1}{(y^2-2xy+1)^{1/2}}\right)\Bigg|_{y=0}$$

A REDUCE version of this is

```
ALGEBRAIC PROCEDURE P(N,X);
   SUB(Y=0,DF((Y**2-2*X*Y+1)**(-1/2),Y,N))/(FOR I:=1:N PRODUCT I)$
```

With this definition, the evaluation of

```
2*P(2,W)
```

would result in the output

$$3W^2 - 1$$

We can of course omit the word ALGEBRAIC in this procedure definition as this is the default type.

In order to allow users relatively easy access to the whole REDUCE source program, system procedures are not protected against user redefinition. If a procedure is redefined, a message

```
*** <procedure name> REDEFINED
```

is printed. If this occurs, and the user is not redefining his own procedure, he is well advised to rename it!

2.19 Commands Used in Interactive Systems

REDUCE is designed primarily for interactive use with a time-shared computer, but naturally it can also operate in a batch processing environment. There is a basic difference, however, between interactive and batch use of the system. In the former case, whenever the system discovers an ambiguity at some point in a calculation, such as a forgotten type assignment for instance, it asks the user for the correct interpretation. In batch operation, it

is not practical to terminate the calculation at such points and require resubmission of the job, so the system makes the most obvious guess of the user's intentions and continues the calculation.

If input is coming from an external file, the system treats it as a batch processed calculation. If the user desires interactive response in this case, he can turn on the flag INT in the file. Likewise, he can turn off INT in the main program if he does not desire continual questioning from the system.

Two commands are available in REDUCE for use in interactive computing. The command PAUSE; may be inserted at any point in an input file. When this command is encountered on input, the system prints the message CONT? on the user's terminal and halts. If the user responds Y (for yes), the calculation continues from that point in the file. On the other hand, if the user responds N (for no), control is returned to the terminal, and the user can input further commands. However, later on he can use the command CONT; and control is then transferred back to the point in the file after the last pause was encountered.

2.20 DEFINE

The command DEFINE allows a user to rename permanently any identifier in the system. Its argument is a list of expressions of the form

```
<identifier> = <number>|<identifier>|<operator>|
                    <reserved word>|<expression>
```

For example,

```
DEFINE BE==,X=Y+Z;
```

means that 'BE' will be interpreted as an equal sign, and 'X' as the expression Y+Z from then on. This renaming is done at the input level and therefore takes precedence over any other replacement declared for the same identifier.

2.21 END

The command END; is used to end external files which are used as input to REDUCE. One of its purposes is to turn off the command echo, which is annoying to a user typing at a terminal. However, it also does some file control book-keeping, and should therefore not be omitted.

If an END command is used in the main program, control is then transferred to LISP.

3. MANIPULATION OF ALGEBRAIC EXPRESSIONS

In this Section, we consider some further system facilities for the processing of algebraic expressions.

3.1 The Expression Workspace

When an assignment of an algebraic expression is made, or an expression is evaluated at the top level, (i.e., not inside a compound statement or procedure) the results of the evaluation are automatically saved in an environment which we shall refer to as the workspace. In actual fact, the expression is assigned to a variable *ANS which is available for further manipulation. In order to input the asterisk as part of the variable name, however, it must be prefixed by an exclamation mark.

Example:

If we evaluate the expression (X+Y)**2 at the top level and next wish to differentiate it with respect to Y, we can simply say

```
DF(!*ANS,Y);
```

to get the desired answer.

If the user gets tired of typing !*ANS, he can use the DEFINE command to introduce another name for the workspace. For example, the statement

```
DEFINE WS=!*ANS;
```

would mean that from then on WS will be recognized as !*ANS in all input.

If the user wishes to assign the workspace to a variable or expression for later use, the command SAVEAS can be used. It has the syntax

SAVEAS <expression><terminator>

e. g. after the differentiation in the last example, the workspace holds the expression 2*X+2*Y. If we wish to assign this to the variable Z we can now say

```
SAVEAS Z;
```

If the user wishes to save the expression in a form that allows him to use some of its variables as arbitrary parameters, the FOR ALL command can be used.

e. g. FOR ALL X SAVEAS H(X);

with the above expression would mean that H(Z) evaluates to 2*Y+2*Z.

3.2 Output of Expressions

A considerable degree of flexibility is available in REDUCE in the printing of expressions generated during calculations. As we mentioned earlier, no explicit format statements are supplied, as these prove to be of little use in algebraic calculations, where the size of output or its composition is not generally known in advance. Instead, REDUCE provides a series of mode options to the user which should enable him to produce his output in a comprehensible and possibly pleasing form.

As we mentioned earlier, an algebraic expression is normally printed in an expanded form, filling the whole output line with terms. Certain output declarations, however, can be used to affect this format. It should be noted, however, that the transformation of large expressions to produce these varied output formats can take a lot of computing time and space. If a user wishes to speed up the printing of his output in such cases, he can turn off the flag PRI using the OFF command. If this is done, then output is produced in one fixed format, which basically reflects the internal form of the expression, and none of the options below apply. PRI is normally on.

With PRI on, the output declarations available are as follows:

3.2.1 ORDER

The declaration ORDER may be used to order variables on output. Thus

```
ORDER X,Y,Z;
```

orders X ahead of Y, Y ahead of Z and X,Y and Z ahead of other variables in expressions which follow, and all ahead of variables appearing in further ORDER assignments, or those not given an order. The order of variables may be changed by a further call of ORDER, but then the reordered variables would have an order lower than those in earlier ORDER calls.

Thus, ORDER X,Y,Z;

```
ORDER Y,X;
```

would order Z ahead of Y and X.

3.2.2 FACTOR

This declaration takes a list of identifiers or functional expressions as argument. FACTOR is not really a factoring command, but rather a separation command. All terms involving fixed powers of the declared expressions are printed as a product of the fixed powers and a sum of the rest of the terms.

All expressions involving a given prefix operator may also be factored by putting the operator name in the list of factored identifiers.

e. g. FACTOR X,COS,SIN(X);

causes all powers of X and SIN(X) and all functions of COS to be factored.

The declaration REMFAC V1,...,VN; removes the factoring flag from the expressions V1 through VN.

3.2.3 Output Control Flags

In addition to these declarations, the form of the output can be modified by switching various output control flags using the declarations ON and OFF. We shall illustrate the use of these flags by an example, namely the printing of the expression

$$X**2*(Y**2+2*Y)+X*(Y**2+Z)/(2*A).$$

The relevant flags are as follows:

(1) ALLFAC. This flag will cause the system to search the whole expression, or any sub-expression enclosed in parentheses, for simple multiplicative factors and print them outside the parentheses. Thus our expression with ALLFAC off will print as

$$(2*X^2*Y^2*A + 4*X^2*Y*A + X*Y^2 + X*Z)/(2*A)$$

and with ALLFAC on as

$$X*(2*X*Y^2*A + 4*X*Y*A + Y^2 + Z)/(2*A)$$

ALLFAC is normally on, and is on in the following examples, except where otherwise stated.

(2) DIV. This flag makes the system search the denominator of an expression for simple factors which it divides into the numerator, so that rational fractions and negative powers appear in the output. With DIV on, our expression would print as

$$X*(X*Y^2 + 2*X*Y + 1/2*Y^2*A^{(-1)} + 1/2*A^{(-1)}*Z)$$

DIV is normally off.

(3) LIST. This flag causes the system to print each term in any sum on a separate line. With LIST on, our expression prints as

$$X*(2*X*Y^2*A$$

$$+ 4*X*Y*A$$

$$+ Y^2$$

$$+ Z)/(2*A)$$

LIST is normally off.

(4) RAT. This flag is only useful with expressions in which variables are factored with FACTOR. With this mode, the overall denominator of the expression is printed with each factored sub-expression. We assume a prior declaration FACTOR X; in the following output . We first print the expression with RAT off:

$$(2*X^2*Y*A + 2) + X*(Y^2 + Z))/(2*A)$$

With RAT on the output becomes:

$$X^2*Y*(Y + 2) + X*(Y^2 + Z)/(2*A)$$

RAT is normally off.

Next, if we leave X factored, and turn on both DIV and RAT, the result becomes

$$X^2*Y*(Y + 2) + 1/2*X*A^{(-1)}*(Y^2 + Z)$$

Finally, with X factored, RAT on and ALLFAC off we retrieve the original structure

```
 2   2                 2
X *(Y  + 2*Y) + X*(Y  + Z)/(2*A)
```

3.2.4 WRITE Command

It is often useful to write a title or comment on output, or name output expressions in a particular way. This is possible in REDUCE by means of the command WRITE. The form of this command is

WRITE <expression>,...,<expression>;

where <expression> may be either an algebraic expression or a string (Section 2.7). Strings are printed on output exactly as given except for any characters which are ignored by the input scanner. Other expressions are evaluated and their value printed. The value of WRITE is the value of its last argument.

The print line is closed at the end of the WRITE command evaluation.

Example:

The following program calculates the famous f and g series [1]:

```
X1:= -SIG*(MU+2*EPS)$
X2:= EPS-2*SIG**2$
X3:= -3*MU*SIG$
F:= 1$
G:= 0$
FOR  I:= 1 STEP 1 UNTIL 10 DO BEGIN
     F1:= -MU*G + X1*DF(F,EPS) + X2*DF(F,SIG) + X3*DF(F,MU)$
     WRITE "F(",I,") := ",F1;
     G1:= F + X1*DF(G,EPS) + X2*DF(G,SIG) + X3*DF(G,MU)$
     WRITE "G(",I,") := ",G1;
     F:=F1$
     G:=G1$
     END;
```

A portion of the output, to illustrate the printout from the WRITE command, is as follows:

```
                ... <prior output> ...

                                  2
F(4) := MU*(3*EPS - 15*SIG  + MU)

G(4) := 6*SIG*MU

                                      2
F(5) := 15*SIG*MU*( - 3*EPS + 7*SIG  - MU)

                                  2
G(5) := MU*(9*EPS - 45*SIG  + MU)
```

... <more output> ...

3.2.5 Suppression of Zeros

It is sometimes annoying to have zero assignments (i.e. assignments of the form <expression> := 0) printed, especially in printing large arrays with many zero elements. The output from such assignments can be suppressed by turning on the flag NERO.

3.3 User Control of the Evaluation Process

In addition to the wide range of output options available to the user, there are two additional flags which control the internal evaluation of expressions. The flag EXP controls expansion of expressions. If it is off no expansion of powers or products of expressions occurs.

When two rational functions are added, REDUCE normally produces an expression over a common denominator. However, if the user does not want denominators combined, he can turn off the flag MCD which controls this process. The latter flag is particularly useful if no greatest common divisor calculations are desired, or excessive differentiation of rational functions is required.

EXP and MCD are normally on.

3.4 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. The system will perform this greatest common divisor check if the flag GCD is on.

A check is automatically made, however, for common variable and numerical products in the numerators and denominators of expressions, and the appropriate cancellations made.

3.5 Numerical Evaluation of Expressions

It is naturally possible to evaluate expressions numerically in REDUCE by giving all variables and sub-expressions numerical values. However, as we pointed out in Section 2.11.1 the user must declare real arithmetical operation by turning on the flag FLOAT. Furthermore, there are no routines for evaluation of the elementary functions COS,SIN etc for arbitrary numerical argument. Finally, arithmetic in REDUCE is not particularly fast.

The user with a large amount of numerical computation after all necessary algebraic manipulations have been performed is therefore well advised to perform these calculations in a FORTRAN or similar system. For this purpose, REDUCE offers facilities for users to produce FORTRAN compatible files for numerical processing.

First, when the flag FORT is on, the system will print expressions in a FORTRAN notation. Expressions begin in column 7. If an expression extends over one line, a continuation mark (X) appears on subsequent cards. After 19 continuation lines are produced, a new expression is started. If the expression printed arises from an assignment to a variable, the variable is printed as the name of the expression. Otherwise the expression is named ANS. Secondly, the WRITE command can be used to produce other programs.

Example:

The following REDUCE statements

```
ON FORT;
OUT FORFIL;
WRITE "C      THIS IS A FORTRAN PROGRAM";
WRITE " 1     FORMAT(E13.5)";
WRITE "       U=1.23";
WRITE "       V=2.17";
WRITE "       W=5.2";
X:=(U+V+W)**11;
WRITE "C      OF COURSE IT WAS FOOLISH
          TO EXPAND THIS EXPRESSION";
WRITE "       PRINT 1,X";
WRITE "       END";
SHUT FORFIL;
OFF FORT;
```

will generate a file FORFIL which contains:

```
C      THIS IS A FORTRAN PROGRAM
 1     FORMAT(E13.5)
       U=1.23
       V=2.17
       W=5.2
       X=U**11+11*U**10*V+11*U**10*W+55*U**9*V**2+110*U**9
      X*V*W+55*U**9*W**2+165*U**8*V**3+495*U**8*V**2*W+495
      X*U**8*V*W**2+165*U**8*W**3+330*U**7*V**4+1320*U**7*
      XV**3*W+1980*U**7*V**2*W**2+1320*U**7*V*W**3+330*U**
      X7*W**4+462*U**6*V**5+2310*U**6*V**4*W+4620*U**6*V**
      X3*W**2+4620*U**6*V**2*W**3+2310*U**6*V*W**4+462*U**
      X6*W**5+462*U**5*V**6+2772*U**5*V**5*W+6930*U**5*V**
      X4*W**2+9240*U**5*V**3*W**3+6930*U**5*V**2*W**4+2772
      X*U**5*V*W**5+462*U**5*W**6+330*U**4*V**7+2310*U**4*
      XV**6*W+6930*U**4*V**5*W**2+11550*U**4*V**4*W**3+
      X11550*U**4*V**3*W**4+6930*U**4*V**2*W**5+2310*U**4*
      XV*W**6+330*U**4*W**7+165*U**3*V**8+1320*U**3*V**7*W
```

```
      X+4620*U**3*V**6*W**2+9240*U**3*V**5*W**3+11550*U**3
      X*V**4*W**4+9240*U**3*V**3*W**5+4620*U**3*V**2*W**6+
      X1320*U**3*V*W**7+165*U**3*W**8+55*U**2*V**9+495*U**
      X2*V**8*W+1980*U**2*V**7*W**2+4620*U**2*V**6*W**3+
      X6930*U**2*V**5*W**4+6930*U**2*V**4*W**5+4620*U**2*V
      X**3*W**6+1980*U**2*V**2*W**7+495*U**2*V*W**8+55*U**
      X2*W**9+11*U*V**10+110*U*V**9*W+495*U*V**8*W**2+1320
      X*U*V**7*W**3
       X=X+2310*U*V**6*W**4+2772*U*V**5*W**5+2310*U*V**4*W
      X**6+1320*U*V**3*W**7+495*U*V**2*W**8+110*U*V*W**9+
      X11*U*W**10+V**11+11*V**10*W+55*V**9*W**2+165*V**8*W
      X**3+330*V**7*W**4+462*V**6*W**5+462*V**5*W**6+330*V
      X**4*W**7+165*V**3*W**8+55*V**2*W**9+11*V*W**10+W**
      X11
C      OF COURSE IT WAS FOOLISH  TO EXPAND THIS EXPRESSION
       PRINT 1,X
       END
```

The number of continuation cards per statement can be modified by the assignment

```
!*CARDNO := <number>;
```

where <number> is the TOTAL number of cards allowed in a statement. *CARDNO is thus initially set to 20.

3.6 Saving Expressions for Later Use as Input

It is often useful to save an expression on an external file for use later as input in further calculations. The commands for opening and closing output files were explained in Section 2.14. However, we see in the examples in Section 3.2 that the standard 'natural' method of printing expressions in not compatible with the input syntax. So to print the expression in an input compatible form we must inhibit this natural style by turning off the flag NAT. If this is done, a dollar sign will also be printed at the end of the expression.

Example:

The following program

```
OFF NAT;
OUT OUT;
X := (Y+Z)**2;
WRITE "END";
SHUT OUT;
ON NAT;
```

will generate a file OUT which contains

```
X := Y**2 + 2*Y*Z + Z**2$
```

```
END$
```

3.7 Partitioning Expressions

It is often necessary to get at parts of an expression during a calculation. REDUCE provides three operators for this purpose. These have proved adequate for all calculations brought to the author's attention, but other commands could be added if there is sufficient demand.

(1) COEFF is an operator which assigns coefficients of the various powers of a kernel. It takes three arguments and has the syntax:

COEFF(<expression>,<kernel>,<identifier>)

If the <identifier> has been previously declared a single dimensioned array, the ith array element is assigned to the coefficient (zero or non zero) of the Ith power of <kernel> in <expression>, up to the maximum dimension of the array.

If the <identifier> is not an array name, a variable with name <identifier><power> is assigned to the coefficient of that power. Only NON ZERO powers are set in this manner, and a message is printed to inform the user of the variables set.

Example:

```
ARRAY X(7);
COEFF((Y**2+Z)**3,Y,X);
```

will cause X(7) to be set to 0, X(6) to 1, X(5) to 0, X(4) to 3*Z and so on until X(0), which would be set to Z**3.

On the other hand, if we said COEFF((Y**2+Z)**3,Y,W);

we would get a message

W6 W4 W2 W0 are non zero

and W6 would be set to 1, W4 to 3*Z and so on.

It is also possible to place the various coefficients generated by COEFF in a particular column of a multi-dimensional array. To do this, the <identifier> in the above symbols should be replaced by an array expression in which the relevant array column is indicated by an asterisk.

For example,

```
ARRAY XX(7,7,7);
COEFF((Y**2+Z)**3,Y,XX(5,*,7));
```

will cause XX(5,7,7) to be set to 0, XX(5,6,7) to 1, XX(5,5,7) to 0 and so on.

The value of COEFF is the highest non-zero power encountered in the expression. Thus in the above examples it would be 6.

(2) NUM and DEN are operators which take a single expression as argument and which return the numerator and denominator of this expression respectively.

E.g., NUM(X/Y**2) has the value X and DEN(X/Y**2) the value Y**2.

4. MATRIX CALCULATIONS

4.1 Preliminary

A very powerful feature of the REDUCE system is the ease with which matrix calculations can be performed. To extend our syntax to this class of calculations we need to add another prefix operator, MAT, and a further variable and expression type as follows:

4.2 MAT

This prefix operator is used to represent n x m matrices. MAT has n arguments interpreted as rows of the matrix, each of which is a list of m expressions representing elements in that row. For example, the matrix

```
(A B C)
(     )
(D E F)
```

would be written as

```
MAT ((A,B,C),(D,E,F))
```

4.3 Matrix variables

An identifier may be declared a matrix variable by the declaration MATRIX. The size of the matrix may be declared explicitly in the matrix declaration, or by default in assigning such a variable to a matrix expression.

e. g. MATRIX X(2,1),Y(3,4),Z;

declares X to be a 2 x 1 (column) matrix, Y to be a 3 x 4 matrix and Z a matrix whose size is declared later by default. All elements of a matrix whose size is declared are initialized to 0.

4.4 Matrix Expressions

These follow the normal rules of matrix algebra as defined by the following syntax:

```
<matrix expression> ::= MAT<matrix description>|<matrix variable>|
                        <scalar expression>*<matrix expression>|
                        <matrix expression>*<matrix expression>
                        <matrix expression>+<matrix expression>|
                        <matrix expression>**<integer>|
                        <matrix expression>/<matrix expression>
```

Sums and products of matrix expressions must be of compatible size otherwise an error will result during their evaluation. Similarly, only square matrices may be raised to a power. A negative power is computed as the inverse of the matrix raised to the corresponding positive power. A/B is interpreted as A*B**(-1).

Examples:

Assuming X and Y have been declared as matrices, the following are matrix expressions

```
Y
Y**2*X-3*Y**(-2)*X
Y+ MAT((1,A),(B,C))/2
```

4.5 Operators With Matrix Arguments

Three additional operators are useful in matrix calculations, namely DET, TP and TRACE defined as follows

4.5.1 DET

The operator DET is used to represent the determinant of a square matrix expression.

e.g. DET(Y**2)

is a scalar expression whose value is the determinant of the square of the matrix Y, and

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

is a scalar expression whose value is the determinant of the matrix

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

4.5.2 TP

This operator takes a single matrix argument and returns its transpose. Its use is obvious.

4.5.3 TRACE

The operator TRACE is used to represent the trace of a square matrix. Its use is also obvious.

4.6 Matrix Assignments

Matrix expressions may appear in the right hand side of assignment statements. If the left hand side of the assignment, which must be a variable, has not already been declared a matrix, it is declared by default to the size of the right hand side. The variable is then set to the value of the right hand side.

Such an assignment may be used very conveniently to find the solution of a set of linear equations. For example, to find the solution of the following set of equations

```
A11*X(1) + A12*X(2) = Y1
A21*X(1) + A22*X(2) = Y2
```

we simply write

```
X := 1/MAT((A11,A12),(A21,A22))*MAT((Y1),(Y2));
```

4.7 Evaluating Matrix Elements

Once an element of a matrix has been assigned, it may be referred to in standard array element notation. Thus Y(2,1) refers to the element in the second row and first column of the matrix Y.

5. ADVANCED COMMANDS

In this Section, we consider several extensions of the basic REDUCE system which add to its power as a problem solving tool.

We begin by introducing a new data type which is needed to extend our syntax, namely the kernel.

5.1 Kernels

REDUCE is designed so that each operator in the system has a evaluation (or simplification) function associated with it which transforms the expression into an internal canonical form. This form, which bears little resemblance to the original expression, is described in detail in Reference [2].

The evaluation function may transform its arguments in one of two alternative ways. First, it may convert the expression into other operators in the system, leaving no functions of the original operator for further manipulation. This is in a sense true of the evaluation functions associated with the operators +, * and / , for example, because the canonical form does not include these operators explicitly. It is also true of an operator such as the determinant operator DET discussed in Section 4.5.1, because the relevant evaluation function calculates the appropriate determinant, and the operator DET no longer appears. On the other hand, the evaluation process may leave some residual functions of the relevant operator. For example, with the operator COS, a residual expression like COS(X) may remain after evaluation unless a rule for the reduction of cosines into exponentials, for example, were introduced. These residual functions of an operator are termed kernels and are stored uniquely like variables. Subsequently, the kernel is carried through the calculation as a variable unless transformations are introduced for the operator at a later stage.

In cases where the arguments of the kernel operators may be reordered, the system puts them in a canonical order, based on an internal intrinsic ordering of the variables. However, some commands allow arguments in the form of kernels, and the user has no way of telling what internal order the system will assign to these arguments. To resolve this difficulty, we introduce the notion of a kernel form as an expression which transforms to a kernel on evaluation.

Examples of kernel forms are:

```
A
COS (X*Y)
LOG (SIN (X))
```

whereas

```
A**B
(A+B)**4
```

are not.

We see that Kernel forms are in fact the 'functional expressions' previously allowed in LET and FACTOR statements.

5.2 Substitutions for General Expressions

All substitutions discussed so far have been very limited in scope, because they involve only replacements for variables and Kernels. These substitutions are very efficient to implement because variables and Kernels are stored uniquely in the system. However, there are many situations where more general substitutions are required, most of which require extensive pattern matching within the expression being evaluated . Consequently, such substitutions cannot be as efficiently implemented as those discussed so far.

For the reasons given in References [3] and [4], REDUCE does not attempt to implement a general pattern matching algorithm. However, the present system uses more sophisticated techniques than those discussed in reference [4]. It is now possible for the equations appearing in arguments of LET to have the form

<substitution expression> = <expression>

where <substitution expression> is ANY expression, subject to the following restrictions:

(i) The operators +, - and / cannot appear at the top level in a substitution expression. This restriction is currently under study to see if it can be removed.

e. g. LET COS(X)**2 + SIN(X)**2 = 1; is NOT allowed.

(ii) The operators + and * can only be used in binary form within substitution expressions.

e. g. LET SIN (X + Y) = SIN(X)*COS(Y) + COS(X)*SIN(Y);

is allowed but a substitution for FN(X+Y+Z) would not be. The system will of course substitute for any expression containing + or * as an n-ary operator by making the appropriate expansion of the binary rule.

(iii) The operator - can only be specified as a unary operator within substitution expressions.

e. g. LET COS(-X)= COS(X) is allowed
but LET COS(X-Y) = <expression> is not.

It should be noted, however, that a rule for COS(X+Y) and one for COS(-X) is sufficient to specify the expansion of COS(X-Y).

Any variables appearing in substitution expressions may be declared arbitrary by the FOR ALL construction

e. g. FOR ALL X,Y LET COS(X+Y)=COS(X)*COS(Y)-SIN(X)*SIN(Y);
FOR ALL X LET LOG(E**X) = X, E**LOG(X)= X,
COS(W*T+THETA(X)) = TAU(X);

It is also possible to limit the range of an arbitrary variable by an extension of the syntax of the IF statement. The relevant form is

IF <boolean expression> LET <substitution list>

To include arbitrary variables in this syntax, we prefix such variables by the operator ARB rather than combine the IF construction with a FOR ALL clause.

e. g. IF ARB X<0 LET F(X)=2*X**2;

As before, after a substitution has been made, the expression being evaluated is reexamined in case a new allowed substitution has been generated. This process is continued until no more substitutions can be made. However, this is sometimes undesirable. For example, if one wishes to integrate a polynomial with respect to X by a rule of the form

FOR ALL N LET X**N = X**(N+1)/(N+1);

one only wants the substitution to be made once. (Otherwise X**2 would become X**3/3 which would then become X**4/12 and so on). This resubstitution can be inhibited by turning off the flag RESUBS (which is normally on).

When a substitution expression appears in a product, the substitution is made if that expression divides the product. For example, the rule

LET A**2*C = 3*Z;

would cause A**2*C*X to be replaced by 3*Z*X and A**2*C**2 by 3*Z*C. If the substitution is desired only when the substitution expression appears in a product with the explicit powers supplied in the rule, the command MATCH should be used instead.

For example,

```
MATCH A**2*C = 3*Z;
```

would cause A**2*C*X to be replaced by 3*Z*X, but A**2*C**2 would not be replaced. MATCH can also be used with the FOR ALL or IF constructions described above.

To remove substitution rules of the type discussed in this Section, the CLEAR command can be used, combined, if necessary, with a FOR ALL or IF clause.

```
e.g.  FOR ALL X CLEAR LOG(E**X),E**LOG(X),COS(W*T+THETA(X));
```

Note, however, that the arbitrary variable names in this case MUST be the same as those used in defining the substitution.

5.3 Asymptotic Commands

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 unnecessary computation. The command LET may be used to do this. For example, if only powers of X up to X**7 are needed, the command

```
LET X**8 = 0;
```

will cause the system to delete all powers of X higher than 7.

This method is not adequate, however, when expressions involve several variables having different degrees of smallness. In this case, it is necessary to supply an asymptotic weight to each variable and count up the total weight of each product in an expanded expression before deciding whether to keep the term or not. There are two associated commands in the system to permit this type of asymptotic constraint. The command WEIGHT takes a list of equations of the form

```
<kernel form> = <number>,
```

where <number> must be a positive integer. This command assigns the weight <number> to the relevant kernel form. A check is then made in all algebraic evaluations to see if the total weight of the term is greater than the weight level assigned to the calculation. If it is, the term is deleted. To compute the total weight of a product, the individual weights of each kernel form are multiplied by their corresponding powers and then added.

The weight level of the system is initially set to 2. The user may change this setting, however, by the command

```
WTLEVEL <number>;
```

which sets <number> as the new weight level of the system. Again, <number> must be a positive integer.

6. SYMBOLIC MODE

6.1 Preliminary

Although REDUCE is designed primarily for algebraic calculations, its source language is general enough to allow for a full range of LISP-like symbolic calculations. To achieve this generality, however, it is necessary to provide the user with two modes of evaluation, namely an algebraic mode and a symbolic mode. To enter symbolic mode, the user types SYMBOLIC; (or LISP;) and to return to algebraic mode he types ALGEBRAIC;. Evaluations proceed differently in each mode so the user is advised to check what mode he is in if a puzzling error arises. He can find his mode by typing

```
!*MODE;
```

The current mode will then be printed as ALGEBRAIC or SYMBOLIC.

If you wish to enter the other mode for a limited time, it is possible to say

```
SYMBOLIC <command>          (or LISP <command>)
```

or `ALGEBRAIC <command>`

At the end of the evaluation, you will be back in the previous mode. For example, if the current mode is ALGEBRAIC, then the commands

```
SYMBOLIC  CAR '(A);
X+Y;
```

will be evaluated in symbolic mode and algebraic mode respectively.

This section assumes that the reader has a reasonable acquaintance with LISP 1.5 at the level of the LISP 1.5 Programmer's Manual [5] or Clark Weissman's LISP 1.5 Primer[6]. Persons unfamiliar with this material will have some difficulty understanding this Section!

Except where explicit limitations have been made, most REDUCE algebraic constructions carry over into symbolic mode. However, there are some differences. First, expression evaluation now becomes LISP evaluation. Secondly, assignment statements are handled differently, as we discuss in Section 6.7. Thirdly, local variables and array elements are initialized to NIL rather than 0. (In fact, such variables and elements are also initialized to NIL in algebraic mode, but the algebraic evaluator recognizes NIL as 0). Fourthly, the delimiters SUM and PRODUCT in the FOR statement (Section 2.11.3) are not defined in symbolic mode. Finally, function definitions follow the conventions of Standard LISP [7].

To begin with, we mention a few extensions to our basic syntax which are designed primarily if not exclusively for symbolic mode.

6.2 General Identifiers

In Section 2.4, we limited identifiers to sequences of letters and numbers. However, it is possible to input any sequence of characters in REDUCE as a name by prefixing non alphanumeric characters by the 'escape character' ! .

e.g. A!(B is an allowed identifier. It will print as A(B.

6.3 Symbolic Infix Operators

There are two binary infix operators in REDUCE intended for use in symbolic mode, namely EQ and . (CONS). The precedence of these operators was given in Section 2.6.

6.4 Symbolic Expressions

These consist of scalar variables and operators and follow the normal rules of the LISP meta language.

Examples:

```
X
CAR U . REVERSE V
SIMP (U+V**2)
```

6.5 Quoted Expressions

Because LISP evaluation requires that each variable or expression has a value, it is necessary to add to REDUCE the concept of a quoted expression by analogy with the LISP QUOTE function. This is provided by the single quote mark '.

e. g. 'A represents the LISP S-expression (QUOTE A)
'(A B C) represents the LISP S-expression (QUOTE (A B C))

Note, however, that strings are automatically quoted in symbolic mode. Thus, to print the string "A STRING", one would write

```
PRINC "A STRING";
```

Within a quoted expression, identifier syntax rules are those of REDUCE. Thus (A !. B) is the list consisting of the three elements A, . and B, whereas (A . B) is the dotted pair of A and B.

6.6 LAMBDA Expressions

LAMBDA expressions provide the means for constructing LISP LAMBDA expressions in symbolic mode. They may not be used in algebraic mode.

Syntax:

<LAMBDA expression> ::= LAMBDA <varlist><terminator><statement>
<varlist> ::= (<variable>,...,<variable>)

e. g. LAMBDA (X,Y); CAR X . CDR Y

is equivalent to the LISP LAMBDA expression
(LAMBDA (X Y) (CONS (CAR X) (CDR Y)))

The parentheses may be omitted in specifying the variable list if desired.

LAMBDA expressions may be used in symbolic mode in place of prefix operators, or as an argument of the reserved word FUNCTION.

6.7 Symbolic Assignment Statements

In symbolic mode, if the left side of an assignment statement is a variable, a SETQ of the right hand side to that variable occurs. If the left hand side is an expression, a function definition is assumed.

e. g. X:=Y translates into (SETQ X Y)

whereas

```
ASSOC(U,V) := IF NULL V THEN NIL
              ELSE IF U EQ CAAR V THEN CAR V
              ELSE ASSOC (U,CDR V)
```

defines the LISP function ASSOC.

MACROs and FEXPRs may be defined by prefixing the assignment by the word MACRO or FEXPR.

e. g. we could define a MACRO CONSCONS by

```
MACRO CONSCONS L := EXPAND(CDR L, 'CONS);
```

Function definitions may also be input in the procedural form discussed in Section 2.18. The procedural type in this case is SYMBOLIC (or LISP). For example, the above definition of ASSOC could be written as

```
SYMBOLIC PROCEDURE ASSOC(U,V);
  IF NULL V THEN NIL
   ELSE IF U EQ CAAR V THEN CAR V
   ELSE ASSOC (U, CDR V);
```

FEXPRs and MACROS may also be input in this manner with the procedural types FEXPR and MACRO respectively.

6.8 Communication with Algebraic Mode

If a function is defined in symbolic mode for use as an operator in an algebraic calculation, it is necessary to communicate this to the algebraic processor. This can be done by using the command OPERATOR. Thus

```
OPERATOR FUN1,FUN2;
```

would declare the functions FUN1 and FUN2 as algebraic operators. This declaration MUST be made in symbolic mode, as the effect in algebraic mode is different.

Furthermore, if you wish to use the algebraic evaluator on an argument in a symbolic mode definition, the function REVAL can be used. The one argument of REVAL must be the LISP prefix equivalent of a scalar expression, e. g., (COS (PLUS X Y)). REVAL returns the evaluated expression in a similar form.

6.9 Obtaining the LISP Equivalent of REDUCE Input

A user can obtain the LISP equivalent of his REDUCE input by turning on the flag DEFN (for definition). The system then prints the LISP translation of his input but does not evaluate it. Normal operation is resumed when DEFN is turned off. Users should note that the LISP obtained is implementation dependent and so will vary from machine to machine.

APPENDIX A

SUMMARY OF THE REDUCE SYSTEM

A.1 Reserved Identifiers

We list here all identifiers which are normally reserved in REDUCE. We include in this list all reserved identifiers given in Section 2 plus all command names and operators initially in the system. The reserved identifiers used in high energy physics calculations (Appendix C) are also included for completeness.

Reserved Words	BEGIN DO ELSE END FOR FUNCTION GO GOTO LAMBDA NIL PRODUCT RETURN STEP SUM TO WHILE
Reserved Scalar Variables	E I
Infix Operators	:= = >= > <= < + - * / ** . SETQ AND NOT OR MEMBER EQUAL UNEQ EQ GEQ GREATERP LEQ LESSP PLUS MINUS TIMES QUOTIENT EXPT CONS
Prefix Operators	ARB COEFF COS DEN DET DF EPS G LOG MAT NUM SIN SUB TRACE
Commands	ALGEBRAIC ARRAY CLEAR COMMENT END FACTOR FOR FORALL GO GOTO IF IN INTEGER LET LISP MASS MATCH MATRIX MSHELL NOSPUR OFF ON OPERATOR ORDER OUT PROCEDURE REAL RETURN SAVEAS SCALAR SHUT SPUR SYMBOLIC VECTOR WEIGHT WRITE WTLEVEL

A.2 Commands Normally Available In REDUCE

Notation: E, E1,...,En denote expressions
V,V1,...,Vn denote variables

The number after the description refers to the Section in which the command is described.

ALGEBRAIC E;	If E is empty, the system mode is set to algebraic. Otherwise, E is evaluated in algebraic mode and the system mode is not changed (6.1)
ARRAY V1<size>, ,...,Vn<size>;	Declares V1 through Vn as array names. <size> describes the maximum size of the array (2.12.2)
CLEAR E1,...En;	Removes any substitutions declared for E1 through En from system (2.16)
COMMENT <any>;	Used for including comments in text. <any> is any sequence of characters not including a terminator (2.8)
CONT;	An interactive command which causes the system to continue the calculation from the point in the input file where the last PAUSE was encountered (2.19)
END <any>;	Terminates files used for input to REDUCE. <any> is any sequence of symbols not including a terminator or the reserved words END, ELSE or UNTIL (2.20)
FACTOR E1,...En;	Declares expressions as factors in output (3.2.2)
FOR	Command used to define a variety of program loops (2.11.3)
FORALL V1,...,Vn <command>	Declares variables V1 through Vn as arbitrary in the substitution rules given by <command> (2.15, 2.17, 3.1 and 5.2)
GOTO V;	Performs an unconditional transfer to label V. Can only be used in compound statements (2.11.4)
IF	Used to define conditional statements (2.11.2 and 5.2)
IN V1,...,Vn;	Inputs the external REDUCE files V1 through Vn (2.14.1)

INTEGER V1,...,Vn;	Declares V1 through Vn as integer variables (2.12.1)
LET E1,...,En;	Declares substitutions for the left hand sides of expressions E1 through En. (2.15 and 5.2). In addition, LET can be used to input differentiation rules (2.17)
LISP E;	If E is empty, the system evaluation mode is set to symbolic. Otherwise, E is evaluated in symbolic mode and the system mode not changed (6.1)
MATCH E1,...,En;	Declares substitutions for the left hand sides of E1 through En when matching of explicit powers is required (5.2)
MATRIX E1,...,En;	Declares matrix variables to the system. The Ei may be matrix variable names, or include details of the size of the matrix (4.3)
OFF V1,...,Vn;	Turns off the flags V1 through Vn (2.12.3)
ON V1,...,Vn;	Turns on the flags V1 through Vn (2.12.3)
OPERATOR V1,...,Vn;	Declares V1 through Vn as algebraic operators (2.6 and 6.8)
ORDER V1,...,Vn;	Declares an ordering for variables V1 through Vn on output (3.2.1)
OUT V;	Declares V as output file (2.14.2)
PAUSE;	An interactive command for use in an input file. When it is evaluated, control is transferred to the user's terminal (2.19)
PROCEDURE	Names a statement for repeated use in calculations. Type specification of procedure precedes the command name (2.18 and 6.7)
REAL V1,...,Vn;	Declares variables V1 through Vn as real (2.12.1)
RETURN E;	Causes a transfer out of a compound statement to the next highest program level. Value of E is returned from compound statement. E may be empty (2.11.6)
SAVEAS E;	Assigns E to the current expression in the workspace (3.1)
SCALAR V1,...,Vn;	Declares variables V1 through Vn as scalar (2.12.1)

SHUT V;	Closes the output file V (2.14.3)
SYMBOLIC E;	Same as LISP E;
WEIGHT E1,...En;	Assigns an asymptotic weight to the left hand sides of E1 through En (5.3)
WRITE E1,...,En;	Causes the values of E1 through En to be written on the current output file (3.2.4)
WTLEVEL V;	Sets the asymptotic weight level of the system to V (5.3)

A.3 Mode Flags in REDUCE

This Section lists the flags which may appear as arguments of ON and OFF. The action of the flag when it is ON is described here, unless stated otherwise. The numbers, as previously, refer to the Section in which the flag is described.

ALLFAC	Causes the system to factor out common products on output of expressions (3.2.3)
DEFN	Causes the system to output the LISP equivalent of REDUCE input without evaluation (6.9)
DIV	Causes the system to divide out simple factors on output, so that negative powers or rational fractions can be produced (3.2.3)
ECHO	Causes echoing of input (2.14.1)
EXP	Causes expressions to be expanded during evaluation (3.3)
FLOAT	Prevents conversion of floating point numbers into the ratio of two integers during evaluation (2.11.1)
FORT	Declares output in a FORTRAN notation (3.5)
GCD	Causes the system to cancel greatest common divisors in rational expressions (3.4)
INT	Specifies an interactive mode of operation (2.19)
LIST	Causes output to be listed one term to a line (3.2.3)
MCD	Causes denominators to be combined when expressions are added (3.3)
MSG	Causes diagnostic messages to be printed (2.15)
NAT	Specifies 'natural' style of output (3.6)
NERO	Inhibits printing of zero assignments (3.2.5)
PRI	Specifies fancy printing for output (3.2)
RAT	An output flag used in conjunction with FACTOR. It causes the overall denominator in an expression to be printed with each factored sub-expression (3.2.3)
RESUBS	When RESUBS is OFF, the system does not reexamine an expression for further substitutions after one has been made (5.2)

A.4 Diagnostic and Error Messages in REDUCE

Diagnostic messages in the REDUCE system are of two types; error messages and warning messages. The former usually cause the termination of the current calculation whereas the latter warn the user of an ambiguity encountered or some action taken which may indicate an error. If the system is in an interactive state, it can ask the user when it encounters an ambiguity for the correct interpretation. Otherwise it will make the most plausible guess, print a message informing the user of the choice made, and continue.

If an error is found during the parsing of the input, the current phrase is reprinted with the place marked where the error was encountered. In some systems, it is then possible to edit the line and correct the error.

For completeness, again, we include messages which can occur in High Energy Physics calculations (Appendix C).

A.4.1 Error Messages

A REPRESENTS ONLY GAMMAS IN VECTOR EXPRESSIONS	A can only be used as gamma 5 in vector expressions
ARRAY TOO SMALL	An array used in COEFF is too small to store all powers of the expression being partitioned
CATASTROPHIC ERROR	If this error occurs, please send a listing of the input and a copy of the output to the author
DIFFERENTIATION WRT <expression> NOT ALLOWED	Differentation with respect to anything but a variable is not permitted
DOT CONTEXT ERROR	A quoted expression in symbolic mode has an incorrect LISP syntax
ELEMENT OUT OF BOUNDS	A reference to an array element outside the declared range has been made
INCORRECT ARRAY ARGUMENTS FOR <name>	Non-numerical index has been used with array or matrix <name>
LARGER SYSTEM NEEDED	This means that your problem requires REDUCE facilities not present in your system
LOCAL VARIABLE USED AS OPERATOR	A local variable has been illegally used in an operator context

MATRIX MISMATCH	Two matrix expressions are not correctly matched for addition or multiplication
MATRIX <name> NOT SET	A matrix has been referenced whose elements are not known
MISMATCH OF ARGUMENTS	Indicates that an operator for which an evaluation procedure has been defined is called with the wrong number of arguments
MISSING ARGUMENTS FOR G OPERATOR	A line symbol is missing in a gamma matrix expression
MISSING OPERATOR	Input error
MISSING VECTOR	An expression encountered in a vector context does not contain a vector
NON-NUMERICAL ARGUMENT IN <expression>	An array or matrix element has been referenced by an non-numerical expression
NON SQUARE MATRIX	An invalid operation on a non square matrix has been requested (e. g., a trace)
NOT YET IMPLEMENTED	Please write to the author if you get this message, enclosing a copy of your input
OPERATOR <name> CANNOT BE ARBITRARY	An arbitrary operator cannot be used in a substitution rule
REDUNDANT OPERATOR	Input error
REDUNDANT VECTOR	A redundant vector has been found in a vector expression
SINGULAR MATRIX	System has been asked to invert a singular matrix
SUBSTITUTION FOR <expression> NOT ALLOWED	An invalid expression has occurred in a substitution statement
SYNTAX ERROR	Incorrect syntax in input. This error occurs only if the system is unable to determine what causes the error
SYNTAX <expression> INCORRECT	A syntax error has been found during the evaluation of <expression>
TOO FEW RIGHT PARENTHESES	Input error
TOO MANY RIGHT PARENTHESES	Input error

TYPE CONFLICT FOR <expression>	An expression has been found in the wrong type context
UNMATCHED INDEX ERROR <list>	Unmatched indices have been encountered during the evaluation of a vector expression
ZERO DENOMINATOR	An expression with a zero denominator has been encountered
0/0 FORMED	The system cannot handle 0/0
<expression> CANNOT BE AN OPERATOR	An operator has been given an invalid name
<expression> INVALID PROCEDURE NAME	A procedure has been given an illegal name
<expression> NOT FOUND	An expression expected by the system (e.g., in a CLEAR statement) was not found
<filename> NOT OPEN	SHUT has been given the name of a file which has not been opened or which is already shut
<identifier> UNDEFINED	An unrecognized command name has been found
<number> TOO LONG FOR FORTRAN	A large integer has been found while preparing FORTRAN output
<type> <variable> USED AS SCALAR	A variable with type <type> has been used in a scalar context
<variable> ALREADY DEFINED AS <type>	A declaration for a previously declared variable has been made
<variable> INVALID or <variable> INVALID IN <statement name> STATEMENT	Incorrect statement syntax
<variable> HAS NO MASS	A variable encountered in an MSHELL declaration has no mass assigned to it

A.4.2 Diagnostic Messages

ASSIGNMENT FOR <expression> REDEFINED	<expression> has been reassigned to a new value
COEFF GIVEN EXPRESSION WITH DENOMINATOR <exp>	COEFF has been given an expression with a denominator which may be a function of the kernel whose powers are being separated
MISSING END IN FILE <name>	The file <name> is not terminated with an END statement

<expression> REPRESENTED BY <expression>	The system has represented the first expression by the second
<name> REDEFINED	An operator or procedure has been redefined
<variable> DECLARED <type>	<variable> has been declared <type>

APPENDIX B

B.1 RUNNING REDUCE ON THE STANFORD PDP-10

B.1.1 Preliminary

REDUCE is stored as a 38K system with filename REDUCE.DMP. It may be loaded by typing R REDUCE. When the system is loaded, it prints a version date and returns an asterisk indicating that it is ready for input.

If you require more core for your job, you can get it by typing

↑C
.CORE <size required><cr>
ST<cr>

You will then be back in REDUCE.

B.1.2 Special Features

B.1.2.1 If you give IN and OUT a variable or dotted pair as argument, the output goes to your disk area. In addition, the reserved identifier L is used to represent the line printer on output.

i. e. OUT L;

sends output to the line printer (i.e., device LPT:).

Input from other devices may be specified by preceding the file name by the device name with the proviso that project-programmer numbers must be quoted. For example, to input a file ANDY from disk area [S,JAM] followed by FOO from DTA2:, you would type

IN '(S JAM),ANDY,DTA2!:,FOO;

B.1.2.2 The altmode character may be used as a terminator. However, if it is used, no results are printed when expressions are evaluated; the semicolon must be used in this case.

B.1.3 A SAMPLE PROGRAM

```
.R REDUCE                                   load the program
*COMMENT A SAMPLE PROGRAM;                  comments are allowed
*X:=(Y+Z)**2;                               set X to (Y+Z)**2
     2              2                       here's the result printed
X := Y  + 2*Y*Z + Z                         because we used a semicolon
                                            as a terminator
*DF(X,Z,2);                                 now differentiate X wrt Z twice
2                                           here's that result

*PROCEDURE FAC N;                           now define the factorial
*       BEGIN INTEGER M,N;                  function
*             M:=1;
*         A:  IF N=0 THEN RETURN M;
*             M:=M*N;
*             N:=N-1;
*             GO TO A
*       END;

*2**FAC 3;                                  we can omit the parentheses
64
*FAC (120);                                 or put them in with unary
                                            operators
<yes, big numbers do work!>
*COMMENT THE FOLLOWING IS USEFUL ONLY IF YOU ARE
       INTERESTED IN SYMBOLIC MODE;
*SYMBOLIC;                                  enter symbolic mode

NIL                                         value returned in symbolic mode
*CAR ('(A));                                compute the CAR of (A)
A                                           here's its value
*ASSOC(U,V):=IF NULL V THEN NIL             now define ASSOC
*             ELSE IF U= CAAR V THEN CAR V
*             ELSE ASSOC(U,CDR V);
*** ASSOC REDEFINED                         REDUCE diagnostic message
ASSOC                                       value from ASSOC definition
*ASSOC ('A,'((B . C) (A . D)));             test ASSOC
(A . D)                                     it works!
*END;                                       return to LISP
***                                         value of END routine
"ENTERING LISP..."                          so that you know
*
```

B.2 RUNNING REDUCE ON THE STANFORD CAMPUS FACILITY IBM 360/67

B.2.1 Preliminary

REDUCE for the Stanford IBM 360/67 computer is stored in symbolic form on a 2314 disk file, and is called into core by a LISP program. However, because of the relatively large size of the program, it is necessary to run REDUCE jobs in the LARGE partition, or overnight.

A minimum REDUCE job with the necessary JCL is as follows:

```
<job card>
/* SERVICE CLASS=L
// EXEC PGM=LISP
//REDUCE DD DSN=SYS3.REDUCE2,DISP=OLD
//LISPOUT DD SYSOUT=A
//LISPIN DD *
 RESTORE(REDUCE)
BEGIN NIL
<REDUCE command>
...
<REDUCE command>
END;
/*
```

Filenames which appear in IN and OUT statements are the ddnames of the relevant files. The description of such files must be specified as part of the JCL for the job. For example

```
//TRAC DD DSN=A123.TR,VOL=SER=SYS15,UNIT=2314,DISP=OLD
```

specifies a file named TRAC for input. It would be referenced in REDUCE by the command

```
IN TRAC;
```

Similarly, to output results onto a file OUT using the OUT command, the JCL for OUT might be:

```
//OUT DD DSN=A123.OUT,VOL=SER=SYS07,DISP=(NEW,KEEP),UNIT=2314,
//         SPACE=(TRK,(5,5),RLSE),
//         DCB=(RECFM=FB,LRECL=80,BLKSIZE=80)
```

The default BLKSIZE specification in REDUCE is 80, so that the files TRAC and OUT discussed above would have that form. However, the user can change that parameter by changing the value of the identifier BLKSIZE!*. This change must of course be made before referencing the file. For example, to input a file INP in WYLBUR CARD format one would say, in REDUCE,

```
BLKSIZE!* := 3520;
IN INP;
```

B.2.2 ERROR RECOVERY

If an error occurs in a REDUCE calculation, the program immediately returns control to the OS supervisor after printing some diagnostic information. It is unfortunately not possible at present to continue a REDUCE job step after such an error has occurred.

B.2.3 Additional Error Messages Possible in REDUCE/360

In addition to the error messages listed in Appendix A it is possible for users to get error messages direct from LISP. The following LISP messages can be expected from time to time:

STORAGE FULL — This means that your problem is too large as stated for your current core partition. The only possible solutions are to run the job in a larger core partition or restate your problem so that less intermediate storage is required.

PUSH DOWN OVERF — This means either that you have defined a recursive substitution (see Section 2.15) or that your problem is so complicated that it exceeds the stack capacity of your system. If you are sure that you haven't done anything wrong, see a system expert about building a system with more stack capacity.

The following messages should NOT occur under normal circumstances. If they or any other strange errors occur, PLEASE SEND A COPY OF THE OUTPUT AND A LISTING OF THE INPUT PROGRAM TO THE AUTHOR.

PSW TRAP

OVER OR UNDERFLOW

FUNCTION NOT DEFINED

B.3 RUNNING DECUS REDUCE ON A PDP-10

B.3.1 Preliminary

At PDP-10 installations with the DEC operating system where DECUS REDUCE is available on SYS:, it may be run by typing

R REDUCE <core size><carriage return>

where <core size> is your core partition size in K words. If <core size> is omitted, the default size of about 40K words is used. Users should note however that the default core size provides a very small workspace, and so more core will be needed for non trivial calculations. If you require more core during the running of your job, you can get it by typing

↑C
CORE <core size><cr>
START<cr>

where <core size> is the TOTAL core required.

When the program is loaded, a message is printed giving the system date. An asterisk is then typed to indicate that the system is ready for input. The program expects REDUCE commands from you. You can however enter LISP by the top level command END;

DECUS REDUCE includes some particular facilities not available in the general version of REDUCE. In addition, its file name formats are designed to conform to DEC conventions. These features are described in the following sections.

B.3.2 File Formats

The REDUCE file handling commands IN,OUT and SHUT expect file names in a form similar to DEC conventions. Thus to read REDUCE commands from the file TEST.INP in your own disk area, you would say

IN TEST.INP;

Input from other devices may be specified by preceding the file name by the device name, with the proviso that project-programmer numbers must be in a list and quoted. For example, to input the file ANDY from disk area [102,304] followed by FOO from DTA2:, you would type

IN '(102 304),ANDY,DTA2!:,FOO;

At installations where a line-printer is available as device LPT:, output may be directed to it by the command OUT L;

REDUCE expects that files input into the system will be written in REDUCE. However, it is also possible to load various LISP formats into REDUCE by specifying the file type in the extension. The extensions currently recognized are as follows:

LAP	Stanford LISP 1.6 LAP file
LSP	Stanford LISP 1.6 file. This differs from a LAP file in that DECIMAL arithmetic is assumed in the LSP file and OCTAL in the LAP file.
SL	Standard LISP file.

All other extensions are currently assumed to be associated with REDUCE files. Conversely, if the above extensions are used, the files must be in the corresponding formats.

B.3.3 Special Characters

The following special characters are allowed on input. On output however they will be printed as the standard character given by the table below:

Standard Character	Optional DECUS Character
:=	←
**	↑

In addition, the <altmode> acts as a <dollar> delimiter (i.e, output is not printed) and also terminates the input line.

The standard DEC I/O control characters are also available. In particular, <rubout> deletes the previous character from the input line, <control>U deletes the whole input line and <control>O suppresses output until the next input prompt.

B.3.4 Debugging Facilities in DECUS REDUCE

A few simple debugging facilities are available in REDUCE for the more experienced programmer. These are as follows:

B.3.4.1 Tracing Functions

A command TR is available in REDUCE for tracing LISP function calls. This command, and the associated commands UNTR, TRST and UNTRST, are used in the form:

TR <function name>,<function name>,...,<function name>;

TR calls the LISP function TRACE, and its output is in exactly the same form. The trace may be turned off by the function UNTR which uses the LISP function UNTRACE.

WARNING:

Because LISP establishes fast links to functions in compiled code once that code has been referenced, it is necessary to inhibit this when tracing is required. TR does this as part of its evaluation sequence, but if tracing is not required until late in a program, the fast links already established by then may nullify the trace. To prevent this, TR should be called with no arguments as the first command in the program.

B.3.4.2 Tracing Assignments in Compound Statements

Useful diagnostic information can often be obtained by observing the values which variables acquire during assignments in particular functions. To do this in REDUCE, one uses the command TRST. There are some restrictions on the function names which appear in the arguments of this function, however. First, the functions must obviously be in the system in symbolic form; compiled functions no longer contain information on the assignment variable names. Secondly, the functions must have a compound statement at their top level.

This particular trace may be turned off by the command UNTRST.

B.3.5 Timing Execution of Commands

The REDUCE command TIME; may be used to time the input and execution of REDUCE commands. Times are printed in milliseconds and represent the time taken since the last call of TIME or since the system was loaded.

B.3.6 A Sample Program

The following shows an example of the interactive use of REDUCE on a PDP-10. Note that the asterisks in the first column are printed by the system to indicate that the program is ready for input and are NOT part of the REDUCE syntax.

```
R REDUCE                                load the program
REDUCE 2 (<system date>)...

*COMMENT A SAMPLE PROGRAM;              comments are allowed
*X:=(Y+Z)**2;                           set X to (Y+Z)**2

      2           2                     here's the result printed
X := Y  + 2*Y*Z + Z                     because we used a semicolon
                                        as a terminator
*DF(X,Z,2);                             differentiate X wrt Z twice

2                                       here's that result
```

```
*PROCEDURE FAC N;                          now define the factorial
*       BEGIN INTEGER M,N;                 function
*             M:=1;
*         A:  IF N=0 THEN RETURN M;
*             M:=M*N;
*             N:=N-1;
*             GO TO A
*       END;

*2**FAC 3;                                 we can omit the parentheses

64

*FAC (120);                                or put them in with unary
                                           operators
<yes. big numbers do work!>

*SYMBOLIC;                                 enter symbolic mode

NIL                                        value returned in symbolic mode

*CAR ('(A));                               compute the CAR of (A)

A                                          here's its value

*ALGEBRAIC;                               return to algebraic mode

*X;                                        evaluate X again

 2           2
Y  + 2*Y*Z + Z                             it's still (Y+Z)**2

*END;                                      return to LISP
ENTERING LISP...                           so that you know

*
```

APPENDIX C

CALCULATIONS IN HIGH ENERGY PHYSICS

NOTATION: In order to allow for the printing of this Manual on printers with limited character sets, we represent Greek characters in this Appendix by their (upper case) English names.

C.1 Preliminary

A set of REDUCE commands is provided for users interested in symbolic calculations in high energy physics. Several extensions to our basic syntax are necessary, however, to allow for the more complicated data structures encountered.

C.2 Operators used in High Energy Physics Calculations.

We begin by introducing three new operators required in these calculations.

C.2.1 .

The . operator is a binary operator used in algebraic mode 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 E0 represents the unit vector (1,0,0,0), (P.E0) represents the zeroth component of the four-vector P. Our metric and notation follows Bjorken and Drell[8]. Similarly, an arbitrary component P may be represented by (P.U). If contraction over components of vectors is required, then the declaration 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)
```

The metric tensor g_{uv} may be represented by (U.V). If contraction over u and v is required, then U and V should be declared as indices.

The declaration REMIND V1...VN $ removes the index flags from the variables V1 through VN. However, these variables remain vectors in the system.

C.2.2 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 identifier (not a number) used to distinguish different lines.

Thus

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

denotes the product of P associated with a fermion line identified as L1, and 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 GAMMA5.

Thus

G(L,A) = GAMMA5 associated with line L.

G(L,P,A) = GAMMA.P*GAMMA5 associated with line L.

$GAMMA_U$ (associated with line L) may be written as G(L,U), with U flagged as an index if contraction over u is required.

The notation of Bjorken and Drell [8] is assumed in all operations involving gamma matrices.

C.2.3 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

```
EPS      = ( +1 if I,J,K,L is an even permutation of 0,1,2,3
   IJKL    ( -1 if an odd permutation
           ( 0 otherwise
```

A contraction of the form EPS p q may be written as EPS(I,J,P,Q),
                            IJuv u v

with I and J flagged as indices, and so on.

C.3 Vector variables

Apart from the line identification identifier in the G operator, all other arguments of the operators in Section C.2 are vectors. Variables used as such must be declared so by the type declaration VECTOR.

e. g. VECTOR P1,P2;

declares P1 and P2 to be vectors. Variables declared as indices or given a mass (Section C.6) are automatically declared vector by these declarations.

C.4 Additional Expression Types

Two additional expression types are necessary for high energy calculations, namely

C.4.1 Vector Expressions

These follow the normal rules of vector combination. Thus the product of a scalar or numerical expression and a vector expression is a vector, as are the sum and difference of vector expressions. If these rules are not followed, error messages are printed. Furthermore, if the system finds an undeclared variable where it expects a vector variable, it will ask the user in interactive mode whether to make that variable a vector or not. In batch mode, the declaration will be made automatically and the user informed of this by a message.

Examples:

Assuming P and Q have been declared vectors, the following are vector expressions

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

whereas P*Q and P/Q are not.

C.4.2 Dirac Expressions

These denote those expressions which involve gamma matrices. A gamma matrix is implicitly a 4 x 4 matrix, and so the product, sum and difference of such expressions, or the product of a scalar and Dirac expression is again a Dirac expression. There are no Dirac variables in the system, so whenever a scalar variable appears in a Dirac expression without an associated gamma matrix expression, an implicit unit 4 x 4 matrix is assumed.

e. g. G(L,P) + M denotes G(L,P) + M*<unit 4 x 4 matrix>

Multiplication of Dirac expressions, as for matrix expressions, is of course non-commutative.

C.5 Trace Calculations

When a Dirac expression is evaluated, the system computes one quarter of the trace of each gamma matrix product in the expansion of the expression. One quarter of each trace is taken in order to avoid confusion between the trace of the scalar M, say, and M representing M * <unit 4 x 4 matrix>.

The algorithms used for trace calculations are the best available at the time this system was produced. For example, in addition to the algorithm developed by Chisholm [9] for contracting indices in products of traces, REDUCE uses the elegant algorithm of Kahane [10] for contracting indices in gamma matrix products.

It is possible to prevent the trace calculation over any line identifier by the declaration NOSPUR.

e. g. NOSPUR L1,L2;

will mean that no traces are taken of gamma matrix terms involving the line numbers L1 and L2.

A trace of a gamma matrix expression involving a line identifier which has been declared NOSPUR may be later taken by making the declaration SPUR.

C.6 Mass Declarations

It is often necessary to put a particle 'on the mass shell' in a calculation. This can, of course, be accomplished with a LET command such as

LET P.P= M**2;

but an alternative method is provided by two commands MASS and MSHELL. MASS takes a list of equations of the form:

<vector variable> = <scalar variable>

e. g. MASS P1=M, Q1=MU;

The only effect of this command is to associate the relevant scalar variable as a mass with the corresponding vector. If we now say

MSHELL <vector variable>,...,<vector variable>;

and a mass has been associated with these arguments, a substitution of the form

<vector variable>.<vector variable> = <mass>**2

is set up. An example of this is given in the following.

C.7 Example

We give 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 [8], Eqs. (7.72) through (7.74).

We wish to compute

$$\text{ALPHA}^2/2\,(k'/k)\,\text{trace}\left(\left(\frac{PF+m}{2m}\right)\left(\frac{E'EKI}{2k.PI}+\frac{EE'KF}{2k'.PI}\right)\left(\frac{PI+m}{2m}\right)\left(\frac{KIEE'}{2k.PI}+\frac{KFE'E}{2k'.PI}\right)\right)$$

where Ki and Kf are the four-momenta of incoming and outgoing photons (with polarization vectors e and e' and laboratory energies k and k' respectively) and pi,pf are incident and final electron four-momenta. Upper case momenta in the above formula are used to indicate contractions of the momenta with gamma matrices. For example, PF = GAMMA . pf.

Omitting the factor ALPHA**2/(2*m**2)*(k'/k)**2 we need to find

$$1/4\ \mathrm{trace}\ ((PF+m)\left(\frac{E'EKI}{2k.pi} + \frac{EE'KF}{2k'.pi}\right)(PI+m)\left(\frac{KIEE'}{2k.pi} + \frac{KFE'E}{2k'.pi}\right))$$

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

```
ON DIV; COMMENT THIS GIVES US OUTPUT IN SAME FORM AS BJORKEN AND DRELL;
MASS KI= 0, KF= 0, PI= M, PF= M; VECTOR E;
COMMENT IF E IS USED AS A VECTOR, IT LOSES ITS SCALAR IDENTITY AS THE
        BASE OF NATURAL LOGARITHMS;
MSHELL KI,KF,PI,PF;
LET PI.E= 0, PI.EP= 0, 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= 0, KI.KF= M*(K-KP), KF.EP= 0, E.E= -1, EP.EP=-1;
FOR ALL P LET GP(P)= G(L,P)+M;
COMMENT THIS IS JUST TO SAVE US A LOT OF WRITING;
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)) $
WRITE "THE COMPTON CROSS-SECTION IS ",!*ANS;
```

This program will print the following result

$$\text{THE COMPTON CXN IS } 1/2*K*KP^{(-1)} + 1/2*K^{(-1)}*KP + 2*E.EP^{2} - 1$$

REFERENCES

[1] Sconzo, P., LeSchack, A. R., and Tobey, R., Symbolic Computation of f and g Series by Computer. Astronomical Journal 70 (May 1965).

[2] Hearn, A. C., REDUCE 2, A System and Language for Algebraic Manipulation, Proceedings of the Second Symposium on Symbolic and Algebraic Manipulation (to be published)

[3] Hearn, A. C., REDUCE, A User-Oriented Interactive System for Algebraic Simplification, Proceedings of the ACM Symposium on Interactive Systems for Experimental Applied Mathematics, held in Washington, D.C., August 1967.

[4] Hearn, A. C., The Problem of Substitution, Proceedings of the 1968 Summer Institute on Symbolic Mathematical Computation, IBM Programming Laboratory Report FSC 69-0312 (1969)

[5] McCarthy, J., Abrahams, P. W., Edwards, D. J., Hart, T. P. and Levin, M. I., LISP 1.5 Programmer's Manual, M.I.T. Press, 1965

[6] Weissman, Clark, LISP 1.5 Primer, Dickenson, 1967

[7] Hearn, A. C., Standard LISP, Stanford Artificial Intelligence Project Memo AI-90 (May 1969)

[8] Bjorken, J. D. and Drell, S. D., Relativistic Quantum Mechanics (McGraw-Hill, New York, 1965).

[9] Chisholm, J. S. R., Il Nuovo Cimento X, 30, 426 (1963)

[10] Kahane, J., Journal Math. Phys. 9, 1732 (1968)