DECUS PROGRAM LIBRARY

DECUS NO.	11-68
TITLE	ALGEBRA - A PROGRAM FOR MANIPULATING LOGICAL EXPRESSIONS
AUTHOR	P. J. Brown and R. C. Saunders
COMPANY	University of Kent at Canterbury Canterbury, Kent, England
DATE	July 21, 1972
SOURCE LANGUAGE	PAL-11

Background

The Algebra program [1] has been implemented for the PDP-11 using the LOWL [2] version of the logic, and the ICL 4130 version of ML/I [3] to do the mapping.

This document describes the version of Algebra which runs under DOS (Disk Operating System) for the PDP11/20.

An earlier version of Algebra is available to run on PDP11's which do not support DOS. As this "stand-alone" version allows interrupt-driven teletype input/output only, and there is normally no problem of limited workspace with Algebra, the DOS version, which allows the use of command strings to assign input/output channels, is recommended.

The following is effectively a re-write of the description of the Algebra program [1] with changes for the I/O facilities under DOS.

The appendix describes the differences between the DOS version and the "stand-alone" program.

Introduction

The ALGEBRA program allows the user to declare a set of objects and then to define a number of operators that can be applied to these objects. The objects are called values. Once the operators and values have been defined the user can investigate their properties by evaluating expressions involving variables, operators and values.

The following is an introductory example, where there are two values, TRUE and FALSE, and the operators are the well-known Boolean operators "implies" and "not". The underlined parts are typed by the computer and the remainder by the console user. A commentary appears to the right of the example.

Introductory example

$RUN ALGEBRA	Bring ALGEBRA into action
ALGEBRA VØØ1	
# KB:<KB:	Results to keyboard < Data from keyboard
VALUES= TRUE FALSE	Declare values.
OPERATOR -	Define an operator. This is the operator "not", which is represented by a minus sign.
UNARY OR BINARY= UNARY	
PRECEDENCE= 100	See below for meaning of this.
-TRUE= FALSE	Define meaning of "not"
-FALSE= TRUE	for all possible values.
OPERATOR HOOK	Define another operator. This is the operator "implies", which is represented by the symbol "HOOK"
UNARY OR BINARY= BINARY	
PRECEDENCE= 50	Precedence has been specified as less than that of "not". This means that if "implies" and "not" are used in the same expression, then "not" is done first, i.e. -A HOOK B is taken as (-A) HOOK B and not as -(A HOOK B)

TRUE HOOK TRUE	= TRUE
TRUE HOOK FALSE	= FALSE
FALSE HOOK TRUE	= TRUE
FALSE HOOK FALSE	= TRUE

Now that the operators and values have been defined, it is possible to evaluate expressions involving them. Any symbol in an expression that has not been defined as a value or an operator is taken as a variable and is enumerated for all possible values. The TABLE statement illustrates this.

```
TABLE A HOOK - (A HOOK B)

A _ _ _B_ _ _ _ _: VALUE

TRUE    TRUE      : FALSE
TRUE    FALSE     : TRUE
FALSE   TRUE      : TRUE
FALSE   FALSE     : TRUE
```

This entire table is typed out by the computer. The last column is the value of the expression for the given values of the variables A and B.

```
TABLE -(P HOOK Q) HOOK(P HOOK -Q)

P _ _ _Q_ _ _ _ _: VALUE

TRUE    TRUE      : TRUE
TRUE    FALSE     : TRUE
FALSE   TRUE      : TRUE
FALSE   FALSE     : TRUE
```

Table typed by the computer.

The TRY statement is an abbreviated form of the TABLE statement, e.g.

```
TRY A HOOK -A(HOOK B)
IS A CONTINGENCY
```

This means that the value of the expression depends on the values of its variables.

```
TRY -(P HOOK Q) HOOK (P.HOOK -Q)
= TRUE                                  This means that the expression
                                        has the same value, TRUE, for
                                        all values of its variables.

↑C                                      end of KB input with
.EN                                     CTRL C EN carriage return
                                        line feed
#↑C                                     # asks for another command
                                        string (i.e. to continue process
. KI                                    with new dataset specifications).
$                                       to exit type CTRL C KI carriage
                                        return
```

Basic Units

The above example should have given the user enough knowledge to use the system for himself, and this he is recommended to do.

The rest of this manual explains more exactly the concepts that have been illustrated by example. Firstly the fundamental units with which ALGEBRA deals, namely symbols, expressions and statements, will be described.

Symbols

A symbol is used to represent the name of a value, operator or variable. A symbol must be either

(a) a name symbol, which is a sequence of one or more letters or digits. The sequence may be arbitrarily long and all characters are significant, but symbols will be truncated to six characters if they appear in tables.

or (b) a punctuation symbol, which is a single character that is not any of the following: a letter, a digit, a comma, a tab, a space, a semi-colon, a left parenthesis, a right parenthesis or the newline character.

For example the following are legitimate symbols:

ABLE,A,1,12A3,+,/,.,VERYLONGNAME

and the following are not:

A B,P.J.B.,(,A+

Expressions

An expression is a series of operators with values or variables as operands. Any symbol that is not the name of an existing operator or value can be used as the name of a variable.

(To be precise the syntax of an expression is, in Backus Normal Form:

```
<expression>::= <expression> <binary operator> <expression>|
<unary operator> <expression>| (<expression>)|<variable>|<value>
```

where <unary operator> etc. are all symbols.)

During the evaluation of an expression, operators of highest precedence are performed first. Subject to this, operators are evaluated from left to right. Parentheses can be used to override precedence and redundant pairs of parentheses are permissible.

Thus if "+" and "*" are binary operators and "*" has the higher precedence then:

```
    A+B*C   is taken as  A+(B*C)
    A+B+C   is taken as  (A+B)+C
  A*B+C*D   is taken as  (A*B)+(C*D)
A*(B+C)*D   is taken as  (A*(B+C))*D
```

The choice of precedence for operators is a matter of convention and no fixed rules can be stated save that unary operators should normally be given higher precedence than binary operators.

Statements

Each line of data fed to ALGEBRA must be a statement. There are four kinds of statement: the OPERATOR statement, the TABLE statement, the TRY statement and the null statement. The first symbol on a line identifies what statement it is.

A comment may be appended to any statement. The comment must be preceded by a semicolon, e.g.

```
OPERATOR  +  ;  THIS  IS  THE  'OR'  OPERATOR
```

If two adjacent name symbols occur within a statement, they must be separated by one or more spaces, commas or tabs. Redundant commas, tabs and spaces are ignored. Thus

```
TRY  (   A,,   +  B, )  ;   XXX
```

is equivalent to

```
TRY(A+B)
```

but

```
TRY   A   AND   B
```

is not the same as

```
TRY   AANDB
```

Similar formatting rules apply to the answers to the questions asked by ALGEBRA, i.e. these may also contain redundant spaces, tabs and commas and they may also have comments appended to them.

Declaration of values

When ALGEBRA is initially entered it asks the question

```
VALUES =
```

At this point the user types a list of different symbols, separated by spaces, commas and/or tabs. These symbols are used thereafter as the value names. They cannot be changed without starting again from scratch.

Note that it is possible to define any number of value names. For example ALGEBRA has proved very useful in examining three-valued logics.

After the values have been declared ALGEBRA proceeds to execute the sequence of statements fed to it. The four different kinds of statement are described below.

The null statement

Null statements are completely ignored and can be used to place comments, for example

```
;TRY DE MORGAN'S LAW
```

The OPERATOR statement

Syntax: OPERATOR symbol

Any symbol that is not the name of a value may be used as the name of an operator. If the operator symbol being defined is the same as an existing operator then, provided the OPERATOR statement is completed without error, the new definition overrides the old. Operators can therefore be redefined. OPERATOR statements need not necessarily be at the start of the data; it is quite legal to add new operators at any time during the use of ALGEBRA.

The OPERATOR statement is followed by a series of data questions-and-answers.

The first question is

UNARY OR BINARY=

and the answer must be either "UNARY" or "BINARY".

The second question is

PRECEDENCE=

and the answer must be an integer in the range [0,999]. The magnitude of the integer does not matter in itself - it is its magnitude relative to the precedence of other operators that counts. For example a precedence of one is the same as a precedence of 998 if the only other operator has precedence 999.

After the above two question-and-answers a series of question-and-answers follows that define the value of the operator for all possible values of its operands. This takes the form

```
var 1 symbol var 1 =

var 1 symbol var 2 =
        .
        .
        .
var 2 symbol var 1 =
        .
        .
        .
```

where var 1 is the first value defined in the answer to the

VALUES =

question, var 2 is the second, and so on. For a unary operator the above questions are abbreviated, as shown in the introductory example.

If the answer to any of the above questions is incorrect, then the message

EH

is output and the question is repeated.

In some uses of ALGEBRA the user may wish to define operators that are undefined for certain values. This can be done, albeit rather tediously, by including an extra value, called UNDEF say, in the list of values.

The TABLE statement

Syntax: TABLE *expression*

A table is output of the values of the expression for all possible values of its variables. The expression must include at least one variable. There is no upper limit on the permitted number of variables in the expression, but if the size of the rows of the table exceeds the width of a line the format of the table is upset.

The TRY statement

Syntax: TRY *expression*

If the expression has the same value, *V* say, for all possible values of its variables, then the result

= *V*

is given; otherwise the result

IS A CONTINGENCY

is given.

Abbreviations

Statements are, in fact, identified only by the first two characters of the initial symbol on a line. Thus OPERATOR may be written OP (or even OPZQP). The same applies to the answers to the "UNARY OR BINARY =" question.

User-defined symbols, however, cannot be abbreviated. Thus, for example, if IMPOSSIBLE is chosen as a value name this must be spelt out in full every time it is used.

Errors

All errors are diagnosed and give rise to a message on the keyboard. These messages should be self explanatory. Except for some errors in answers to questions, which cause the question to be repeated, all errors cause the current statement to be abandoned, and the next one taken.

Interface with DOS

Finally the interface between ALGEBRA and DOS will be described.

Entry and exit

ALGEBRA is entered by the command

```
$RUN ALGEBRA
```

The N and final A in this command may be omitted because only the first two characters of monitor commands, and the first six of filenames are significant. The separator between the two words may be a comma and/or a space.

ALGEBRA then requests a command in CSI format. ([4] § 3.4.1) by printing a # on the keyboard. The user types a command with one output dataset specification and at most three input dataset specifications,

e.g. `KB:<DT1:ALGDAT,PR:,KB:`

which means that input is taken from the file named ALGDAT catalogued under the user's identifier on dectape unit 1 and output is sent to the keyboard; at the end of the ALGDAT file, the current process is continued (i.e. all previously defined value names and operators remain in existence), but data is taken from the fast paper tape reader, results still going to the teletype keyboard; finally data is taken from the keyboard with results sent to the keyboard.

Note that it is possible to send results to disc with the command

```
#RCS<DT1:ALGDAT,KB:            (1)
```

or dectape with

```
#DTØ:PJB<DT1:ALGDAT,KB:        (2)
```

In (1), a dataset named RCS is created on the disc (DFØ:) and in (2), a dataset PJB on the dectape loaded on unit Ø.

When results are sent to devices other than the console (KB:), all input lines and questions-and-answers are also sent to the device, so that the complete process may be listed later using PIP [4].

Questions and answers

In the non-conversational case, all questions-and-answers are compulsory, and if a question is not matched after ignoring all spaces and tabs, the message

```
***UNMATCHED QUESTION - OFFENDING LINE IS :-
```

appears on the console, and data is then taken from the next input stream, if any. If there is no more input, ALGEBRA asks for another command with a #

End of data on datasets

For disk and dectape files, end of data is signified by the end of file, but a problem arises with paper tapes read from the console reader (PT:) and with data typed in at the keyboard (KB:),

To signify end of data type the monitor command

ENd $\begin{Bmatrix} \text{KB} \\ \text{PT} \end{Bmatrix}$ carriage return linefeed

To give this command the user must first type CTRL C to set the monitor in listening mode. If the argument is omitted, KB is assumed.

Breaks

To break ALGELRA, press CTRL C which sets the monitor in listening mode. The user may then type one of the following four commands:

(1) a BEgin command will restart ALGEBRA from the beginning, and all operators will be lost;

(2) a WAit will return to command status and the current process may then be continued with a COntinue;

(3) a KIll will close all open files and removes ALGEBRA from core, ready for the next program;

(4) a REstart will return to command status if no values have yet been defined; otherwise the program asks for another line of input, with all current values and operators saved.

Continuing the process and exiting

When ALGEBRA has completed the actions specified by the command string, it returns to command status. Three different actions may be taken by the user; to continue the run keeping all current operators and value names in existence, type in a command in CSI format as before specifying new datasets; to start ALGEBRA from the beginning deleting all current data, type 'CTRL C BEgin carriage return'; to call in another program type 'CTRL C KIll carriage return', followed by GEt or RUn and the new program name.

APPENDIX

<u>The "stand-alone" version of Algebra</u>

A completely self contained Algebra program is available in two forms on paper tape:

1) PAL source in ASCII code.

2) absolute loader format.

<u>Differences from DOS version</u>

1) Teletype input/output only.

   The low speed paper tape reader and punch may be used, and there is no problem with read-in speed because type-ahead is allowed.

2) ← is backspace, not 'rubout'

   ! is line erase, not CTRL U.

   . alone on a line signifies end of data.

   CTRL G is the break signal. If this occurs before values have been declared, an exit is made; otherwise all breaks cause ALGEBRA to ask for another line of input, with previously defined values and operators still in operation.

3) The message

   ILLEGAL INPUT CHARACTER

   is printed if an undefined character is input from the keyboard.

EDITOR'S NOTE:

1. The "stand-alone" version on paper tape is <u>not</u> available from DECUS. Please contact the author for further information.

2. ML/I <u>is</u> available from DECUS as DECUS NO. 11-69.

Acknowledgement

This implementation of ALGEBRA was supported by a Science Research Council Grant to investigate the implementation of machine-independent software using the ML/I macro processor.

References

[1] Brown, P.J. and Lowe, J.D. A computer program for symbolic logic. Bulletin Inst. Maths Applics 7,11 (Nov. 1971)

[2] Brown, P.J. Implementing software using the LOWL language. Computing Laboratory, University of Kent at Canterbury, 1971.

[3] Brown, P.J. Using a macro processor to aid software implementation. Comput. J. 12.4 (Nov. 1969), 327 - 331.

[4] PDP11 Disk Operating System Monitor, Programmer's Handbook, Digital Equipment Corporation, 1971