MAD AT MICHIGAN

its function & features

by BRUCE W. ARDEN, BERNARD A. GALLER,
and ROBERT M. GRAHAM, University of Michigan, Ann Arbor

The Michigan Algorithm Decoder (MAD), in operation since February, 1960, was developed for the specific purpose of training large numbers of university students and handling the large volume of university research problems. The primary motivation for writing this rapid translator may be traced directly to the special environment of a university computing center.

A large university should not operate its computing facility as a closed shop since the university's role is educational; moreover, it cannot operate in such a fashion since the users, students and staff, may number in the thousands. The computing implications of these facts are that large numbers of relative novices will have direct access to the machine. Like it or not, these users will work at the level of an algebraic source language—often making many compilations on programs which are not destined to become long-run production programs. What is required in this environment (which is not necessarily peculiar to universities) is an extremely rapid translator accepting a source language which has a minimum number of restrictions. MAD was written to fulfill these requirements. ALGOL 58 provided the basic pattern for the language and to the extent that ALGOL 58 is like ALGOL 60, MAD is an ALGOL translator. The design criteria were satisfactorily achieved although, undeniably, the permitted generality of expression reduced, in some instances, the object program efficiency as compared to that produced from restricted source languages.

MAD translators now exist on the IBM 704, 709, and 7090, and a recent compilation of 785 statements took one minute on the 7090, producing approximately 10,000 machine instructions. In another occasion, using the Bell Monitor System on the 7090, MAD compiled 84 programs in 20 minutes. (MAD is also available as part of the FORTRAN Monitor System.) The translator itself contains about 16,000 words, of which several hundred are diagnostic comments. Before we consider the reasons for the translation speed, we shall describe some of the features of the MAD language. Symbols consist of up to six alphanumeric characters, the first of which is a letter. Function names consist of a symbol and a terminating period, such as SIN., SQRT., and so on. Constants, as in FORTRAN, have such forms as 1.2E-3, .006, -1E7, etc., except that one may write the Boolean constants 1B and 0B (representing *true* and *false*, respectively), and alphabetic constants, such as ABC or $=$ (up to six Hollerith characters enclosed in dollar signs). Arithmetic expressions are formed in the usual way, using +, −, *, /, .ABS., and .P., with the last two meaning "absolute value" and exponentiation, respectively. Also available are the relations $<$, $\leq$, $=$, $\neq$, $>$, and $\geq$, represented by .L., .LE., .E., .NE., .G., and .GE., respectively, and $\wedge$, V, $\neg$, $\supset$, $\equiv$, and exclusive *or*, represented by .AND., .OR., .NOT., .THEN., .EQV. and .EXOR., respectively. Thus, one might write the Boolean expression

```
X .P.3 .L. Y .P.3 .AND. I .NE. J .OR.X * Y .G. 1
```

to represent the expression

$$X^3 < Y^3 \wedge I \neq J \text{ VXY } > 1$$

The mode of arithmetic of each variable and function (floating point, integer, Boolean, etc.) is declared by exception. For example, in MADTRAN, the FORTRAN to MAD translator which was written in MAD, all variables were of integer mode, since Hollerith constants are also of integer mode. The statement

```
NORMAL MODE IS INTEGER
```

caused all variables to be integer, and if there had been any floating point variables, they would have been so declared. There also exist programs in which the normal mode is Boolean! Mixed expressions are allowed whenever ordinary mathematical notation would allow it.

Besides the ordinary substitution statement, which has the form

```
X = (-B + SQRT.(B*B-4.*A*C))/(2.*A)
```

there are many other statements. The most useful statements are the simple and compound conditionals and the two forms of the iteration statement. These are illustrated by the following:

(a) **The Simple Conditional**

```
WHENEVER I .G.J .AND. X .E. 3, TRANSFER TO
BETA(I)
```

(b) **The Compound Conditional**

```
WHENEVER I .G. J .AND. X .E. 3

        C(I) = B(J)
        X = 2

        TRANSFER TO BETA(I)
OR WHENEVER I .G. J .AND. X .E. 2

        C(I) = B(J) + X
        X = X - 1
```

```
OTHERWISE
        C(I) = B(J) - X
        X = X - 1

END OF CONDITIONAL
```

(c) **The Iteration Terminated by a Boolean Expression**

```
THROUGH A, FOR I = BZERO, F.(I) - I,
.ABS.F.(I) .L. EPSLON
```

This is interpreted as: Set I = BZERO, then test the Boolean expression .ABS.F.(I) .L. EPSLON. If *true*, the scope of the iteration is not executed at all. If *false*, it is executed, then I is incremented by F.(I) − I (in effect, I is replaced by F.(I) in this example) and the Boolean expression is tested again.

(d) **The Range of Values Iteration**

```
THROUGH A, FOR VALUES OF Q = 50, 25, 10, 5, 1
```

In this case the scope is executed for each of the listed values (which could be arbitrary expressions) in turn.

While the dimension statement in MAD causes a fixed block of storage to be set aside for each array, the number of dimensions (i.e., the number of subscripts), the range of each subscript, and the base point of the array within the block are stored as ordinary values of a vector and may all be modified or set during execution. Format information, similar in form to that used in FORTRAN, can likewise be modified or set at execution time. On the other hand, all of this information may be pre-set by a general declarative statement which can pre-set values of any array or vector. Input-output statements which make use of a symbol table and require no format information, such as

```
READ DATA
```

have recently been added to the 709/7090 version, as well as the ability to define new operations into the language.

There are also facilities for defining internal or external functions. The latter correspond to the FORTRAN *subroutine*, while the former are similar to the ALGOL *procedure*, in that they are defined within the body of the calling program and variables not listed as arguments are common to both programs. It is possible to define recursive functions, also, and still other statements provide a *push-down list* mechanism. While multiple subscripting is handled by a set of standard subroutines, the user may provide his own storage function for each array, if he chooses, by designating the name of the new function. This function, whose definition program may be written in MAD, may then allow one to store one-half of a symmetric or upper triangular array, only non-zero entries, etc.

There are many other features in MAD, such as the EQUIVALENCE, ERASABLE, and PROGRAM COMMON declarations, and so on, but let us return to the question of the reason for the speed of translation. Probably the greatest single factor in this speed is the minimization of the use of the magnetic tapes. In the 709/7090 version, programs containing under 200 statements require no tape movement other than reading the source program from the input tape and writing the translated object program onto the output tapes. (The MAD translator is itself one record on the master tape.) Longer programs require intermediate tape storage, but in a highly buffered way. Another factor is the internal structure of the translator, with a heavy dependence on just a few tables of information. Also the statement structure is such that statement types are discernable without an analytic scan. A final and, perhaps, second order reason is that the group writing the translator was small—three people. As a result the intra-group communication was good, and the program inefficiencies which are so often introduced in developing large, group-effort programs were held to a minimum. The writing of the 704 MAD translator took 2 man-years, and the rewriting for the 709/7090 took about 3 man-months.

The use of this translator has made it possible to assign rather formidable problems to students. In a one-semester course assuming no computer experience, the following problems have been assigned (each one as the last problem in a different semester). (1) Symbolic differentiation, (2) An interpretive program for a simplified MAD–like language, (3) An assembly program for a language much like FAP for the 709/7090, (4) The complete scan and decomposition of arithmetic expressions from MAD. Usually 30-50% of the 100 or more students in the course have the program checked out by the end of the semester. The following is a portion of a solution to the analytic differentiation problem. It deals with the input expression, and might be found in any of the problems described above. It is included here to illustrate some of the symbol manipulation features of the language. Problems involving evaluation of algebraic expressions would appear much more conventional than this.

```
R       FUNCTION CONST. THIS BOOLEAN
        FUNCTION
R       RETURNS A VALUE OF 1B (TRUE)
        IF ITS ARGUMENT IS
R       INCLUDED IN THE SET OF CONSTANTS
        0-9,A-Z (except X)
R       AND OB (FALSE) OTHERWISE.
        INTERNAL FUNCTION (Z)
        BOOLEAN CONST.
        ENTRY TO CONST.
SCON1 THROUGH SCON1, FOR B =
        1, 1, Z .E. TAB(B) .OR. B .E. 36
        WHENEVER B .L. 36
                FUNCTION RETURN 1B
        OTHERWISE
                FUNCTION RETURN 0B
        END OF CONDITIONAL
        END OF FUNCTION
        VECTOR  VALUES  TAB(1)=$0$,$1$,$2$,$3$,
                $4$,$5$,$6$,$7$,$8$,$9$,
1               $A$,$B$,$C$,$D$,$E$,$F$,$G$,$H$,$I$,
                $J$,$K$,$L$,$M$,$N$,
2               $O$,$P$,$Q$,$R$,$S$,$T$,$U$,$V$,$W$,
                $Y$,$Z$,
R       FUNCTION P. THIS FUNCTION ASSIGNS A
R       PRECEDENCE VALUE TO THE OPERATORS.
        INTERNAL FUNCTION (Z)
        ENTRY TO P.
        WHENEVER Z .E. $.P.$
                PREC=8
        OR WHENEVER Z .E. $-U$
                PREC=7
        OR WHENEVER Z .E.$/$ .OR. Z .E. $*$
                PREC=6
        OR WHENEVER Z .E.$-$ .OR. Z .E. $+$
                PREC=5
        OR WHENEVER Z .E. RTEND .OR. Z .E. LFTEND
                PREC=4
        OTHERWISE
                PREC=3
        END OF CONDITIONAL
        FUNCTION RETURN PREC
        END OF FUNCTION
```