Look Out for Guaranteed Properties

by Winfried Gerum

· *Winfried Gerum*

We all strive for reliable software, or at least we should. But we need reliable primitives and reliable methods. It is up to the MUMPS Development Committee (MDC) and to the implementors to provide the basic primitives MUMPS programmers use and we expect these primitives to be portable and reliable. But do they really act as they are supposed to do? As for the MDC, its primitives often do, but sometimes not.

Unusual but Very Clever: The Modulo Operator

For example, the Committee did a fine job defining the "#" operator. In MUMPS that operator is consistent with the mathematical concept of residue classes with the implication that the behavior with negative values is reasonable. In other languages a modulo operator—if available—usually gives different results (for negative operands) which are easier (faster) to calculate, but more difficult to use. If you have a date in $HOROLOG format, then date#7 indicates the day of the week even if date is less than 1. With time arithmetic, the modulo operator gives the proper time of the day even for negative time values. That guaranteed property of the modulo operator is very useful. But be careful if you port from other languages to MUMPS and vice versa. In other languages the modulo operator (or function) is defined in a different way, such that **a\b*b+(a#b)=a** is always true for all values of a and b (b not equal to 0). In MUMPS that proposition holds just for **positive** values of a and b. (See endnote.)

Arithmetic—Can We Do It on Computers?

Numbers generally are a problem in MUMPS, since there is not much really guaranteed. You don't believe that? Try to get the value of $LENGTH(1/3*3). That expression is all standard MUMPS, but the result is somewhat random. Is that irrelevant to everyday problems? It is not! Look at the places where the standard specifies 'intexpr'. Integer expression discards everything after the decimal point. Therefore it makes a difference whether you have $JUSTIFY(X,3.000000000001) or $JUSTIFY(X,2.999999999978). The former acts as $J(X,3), while the latter is the same as $J(X,2). In a similar way, it makes a difference whether you have IF 0 or IF .000000000031. The latter condition looks very much like zero but here it acts as one. So, small effects may result in big differences.

Certainly nobody will write down such crazy numbers. But they may be the result of some arithmetic. In standard MUMPS arithmetic has few guaranteed properties. The standard is not exactly specific about arithmetic. Nor do all implementors do a good job: try simple code such as

```
S X=1 F I=0:1 W !,X S X=X*2
```

to get all powers of two your MUMPS can handle. Some implementations yield proper values until they terminate with some error message. But other implementations yield values ending with zeroes despite that a power of two never ends with a zero!

One day I tried that code on a very popular implementation. The result was an endless loop with occasional negative numbers. Even some published versions of the square root algorithm such as

```
SQRT(X) Q:X<0 "" Q:'X 0
        N A,B
        S A=1+X/2
        F S B=A,A=X/B+B/2 Q:A'<B
        Q B
```

are not without problems: careful analysis will show that with "exact arithmetic" the terminating condition will never become true for most values of X. If that algorithm works properly on many MUMPS implementations, it is not because of a good job implementing arithmetic. It is that there are many compensating errors that make this terminating condition work as desired. It should be no surprise if you someday get an implementation that loops endlessly calculating $$SQRT(2) with the above algorithm.

If you want to write portable and reliable software, be very careful about what is actually guaranteed.

Bad Guys Are Everywhere

Not only implementors but ordinary MUMPS programmers often are careless about what is guaranteed and what is not. Some glitches:

```
SET X=$E(X,2,255)
```

or

```
SET X=$E(X,2,999)
```

or

```
SET X=$E(X,2,2048)
```

may delete the first character of any string X on many systems, but not all. Standard MUMPS guarantees a minimum of 255 characters for the maximum string length. But in fact it does not specify a maximum for the maximal string length. So the only proper code is

```
SET X=$E(X,2,$L(X))
```

(if `SET $E(X)=""` is not available). Another frequent error is the replacement of code such as

```
IF X="a"!(X="b")!(X="c")
```

by

```
IF "abc"[X
```

or

```
IF $FIND("abc",X)
```

the latter codes are shorter, but equivalent only if it is guaranteed that $LENGTH(X)=1. Otherwise the latter conditions may be true if `X=""` or `X="ab"` or `X="bc"`.

$HORRORLOG

Occasionally, people use $HOROLOG values as a single index (global subscript) with the idea that this will result in a chronological ordering. At first glance, it will do so: from 18–May–1868 through 15–October–2114 the first $PIECE of $HOROLOG is 5 digits and from 2:46:40 AM through midnight the second $PIECE is 5 digits, too. As long as one stays within these limits one gets chronological ordering indeed. But the resulting software is not reliable: as soon as somebody works past midnight, the ordering is messed up. When using $HOROLOG values, split them up into two numeric indices that will always do the job.

Guess What the Current Century Is

Frequently, people write "19" for the current century because as long as they can remember, the current century has been 19. But it has not been that way forever and will not long remain thus. When using $HOROLOG values in sorting, either use

```
SET KEY=$H,KEY=$P(KEY,",",2)
*.00001+KEY
```

or split them up into two numeric indices

```
SET KEY2=$H,KEY1=+KEY2,KEY2=$P
(KEY2,",",2)
```

to get the job properly done.

Be Clever but Not Too Clever

A comparison of numeric values

```
IF A'=B
```

may be written a bit shorter as

```
IF A-B
```

and

```
IF $IO'=DEV
```

can be replaced by the same reasoning with

```
IF $IO -DEV
```

which works on some systems where $IO yields an integer. But the standard does not require $IO values to be integers. That piece of code is not portable.

$DATA

Often code is written as

```
IF '$DATA(VAR) SET VAR="DEFAU
LT-VALUE"
IF VAR>LIMIT . . .
```

The idea is that if VAR is not defined, assigning a value avoids termination of the routine with an <UNDEF> error. In fact, this error is not avoided with this code unless it is guaranteed that `$DATA(VAR)` cannot yield a value of 10. Instead, if we write

```
IF $DATA(VAR)[0 . . .
```

we will never have a problem. If you write extrinsic functions, remember: when you write a function you never know whether parameters are passed by value or by name. Therefore, in code written as

```
FUN(A,B) IF '$DATA(B) SET B=
"default" . . .
```

you expect that B might be undefined because someone called the function with `$$FUN(X)`. But the situation

```
KILL Y WRITE $$FUN(.X,.Y)
```

produces an undefined `B` also and there is no way to tell from within the function how it has been called. Moreover,

```
KILL Y SET Y(1)="" WRITE $$FUN
(.X,.Y)
```

produces a `B` with `$DATA(B)=10` to cause trouble.

How to Stop Worrying and Start Programming

Never take superficial properties for guaranteed properties. If you write software, look at what the MUMPS standard says and read your documentation to know what is true on your MUMPS system. Try to make your own software reliable by being specific about what you want to guarantee.

Endnote

The definition of modulo varies widely between languages:

```
X#Y = X-([X/Y]*Y)        IF Y'=0
   divide by zero error  IF Y=0
   where [a] = the largest integer '>a
   Ada
   BASIC
   Common Lisp
   FORTRAN 90 (MODULO)
   MUMPS

X#Y = X-([X/Y]*Y)        IF Y>0
   divide by zero error  IF Y=0
   undefined error       IF Y<0
   where [a] = the largest integer '>a
   Modula-2
   Pascal

X#Y = X-([X/Y*Y)         IF Y'=0
   0                     IF Y=0
   where [a] = the largest integer '>a
   PL/1

X#Y = X-([X/Y*Y)         IF Y'=0
   X                     IF Y=0
   where [a] = the largest integer '>a
   APL

X#Y = X-([X/Y*Y)         IF Y'=0
   divide by zero error  IF Y=0
   where [a] = the integer part of a,
      rounded towards 0
   FORTRAN 77
   FORTRAN 90 (MOD)
   Lisp

X#Y = machine-dependent
   C

X#Y = implementation-specific
   Prolog
```

—*Editor*

Winfried Gerum is president of Winner Software, a company dedicated to serving the MUMPS community with tools, consulting, and related services. His first company developed medical software with MUMPS. At the end of 1991, he founded his current company. He has been active in the MUMPS community for many years, contributing articles to journals and proposals to the MDCC-Europe.

ESI offers a software training alternative. Computer Based instruction is much less expensive than course fees, and eliminates travel time and expenses. Perfect for individuals or groups. . . the more they're used, the more you save.

SAVE TIME AND TRAINING $$

CBI's available on floppy disk and magtape:

- Introduction to MUMPS
- Intermediate MUMPS Programming
- Advanced MUMPS Programming
- Introduction to File Manager
- Intermediate File Manager
- Advanced File Manager
- WORD MANAGER Tutor
- VAX DSM System Management
- VAX/VMS Concepts
- DataTree System Management
- DCL Programming
- MSM-PC System Management

ESI

On site and lecture/workshop courses also available.
EDUCATIONAL SYSTEMS, INC.
211 Vaughn Hill Rd., Bolton, MA 01740
Tel. 508-779-2914 FAX 508-779-6092

Moving?

Let us know! Contact us at:

MUMPS Users' Group/M Technology Association
1738 Elton Road, Suite 205
Silver Spring, MD 20903
Phone: (301) 431-4070
Fax: (301) 431-0017

Please include your MUG ID number or mailing label from *MUMPS Computing* along with your new address.