An Open Letter

From Fateman to Veltman

Dear Dr. Veltman:

The talk you gave at the First European Conference on Computational Physics gives a most interesting (and I assume intentionally provocative) view of symbol manipulation programs, and for that matter, higher level programming languages in general.

I was wondering to what extent your talk, and the corresponding article in *Computer Physics Communications* (3 supplement, (1972) 75-78) aroused controversy, and if any rebuttal was prompted. I have been meaning to bring up some of the issues you raised, since a recent acquaintance was discouraged from working in the field of algebraic manipulation, at least in part by your pessimism.

First, I would like to state several points of agreement between us.

(a) Human ingenuity is usually very important in solving difficult problems.

(b) Many problems are computationally too difficult for practical solution by man *or* computer.

(c) There is a range of problems which are solvable in a practical (time or space limited) sense only when their "solutions" are stated using a suitable data structure or suitable algorithm. Thus, for example, a machine-language structure might be better than a LISP structure.

(d) Attempts to put together algebraic manipulation programs without a view toward applications and efficiency are not of much interest. (As you say, "there are probably more programs than problems. And it is a sad thing that most of these programs cannot solve any of the problems.")

Secondly, I would like to "agree to disagree" on several matters of secondary importance.

(a) I believe that formatting of output is more important than you seem to feel. Certainly there is no great difference between A**2 and A^2 but there is a significant barrier to comprehension of larger expression when they appear in computer notation, and the form of the answer is non-trivial. For example, a paper on my desk tells me that

```
R(x)=(1/sqrt(x)*atanh((sqrt(2/pi*x)*
    sum((-1)**n*x**n/n!/(2*n+1)/2**n,n,0,inf))**2
  = 2/pi-(2*pi-8)/(3*pi**2)*x+O(x**2).
```

Personally, I would rather see that as the following computer-generated display:

$$R(x)=\{\frac{1}{\sqrt{x}}\tanh^{-1}(\sqrt{\frac{2}{\pi}x}\sum_{n=0}^{\infty}\frac{(-1)^n x^n}{n!(2n+1)2^n})\}^2$$

$$=\frac{2}{\pi}-\frac{2\pi-8}{3\pi^2}x+O(x^2).$$

Further research in this area is certainly fun, and has some benefits.

(b) I believe that assembly language programs may be better than higher-level language programs from a computer utilization standpoint, but if one looks at the total situation of human utilization, one's perspective changes. A well-designed language is of great aid in solving problems, and the time saved in programming may outweigh any increased computational costs. I generally disagree with your statement, "Even more extreme, it is my opinion that in the end higher level languages such as ALGOL 68 and LISP do more harm than good. They cripple the computer. These things appear to me as pencils with an engine attached to them to move the pencil." Some evidence to this effect follows.

As for efficiency in data structures, many modern programming languages provide for manipulation of data in arbitrary units, and with arbitrary interpretations. There is no inherent reason for such languages to deal with "full words" or other wasteful structures.

(c) The importance of applications of algebraic manipulation systems is difficult to evaluate at present, and as such systems become more integrated into the every-day bag of tricks of applied mathematicians and scientists, I think their importance will grow. Bibliographies of "published results" are some evidence of usefulness, although even such lists are difficult to evaluate.

(d) You emphasize the importance of special purpose systems for special problems. Certainly this is an important approach to solving a problem. Speaking to a somewhat different constituency, general-purpose systems have a range of representations for data, and a range of algorithms (greatest-common-divisor, factorization, integration, arbitrary-precision arithmetic, etc.) which *may* solve some of the same problems as the special purpose systems, although slower. Yet because of their greater applicability, the general purpose systems make the entry-level to algebraic problem-solving on a computer more approachable.

I believe the crux of your argument is that the question of efficiency will make some problems solvable *only* in specially designed machine-language systems. This is certainly true now, and will be true for many problems. But I disagree that this is inherent in the use of higher-level languages. Certainly a poorly written assembly-language program can be wasteful, and if, as is often the case, 90% of execution time is spent executing 5% of the code, a good high-level language system will have (if necessary) a collection of machine-language "5%'s". The larger efficiencies of being able to code more sophisticated algorithms more conveniently in high-level languages is hard to quantify, but is suggested below.

In your paper you appear to demolish that workhorse problem, the "F and G Series" by indicating that it is trivial to write ~~a~~ a program in Fortran to solve the problem, and "our man needs to know only Fortran and can use the computer of the grocery around the corner."

In fact, having taken your advice to "think a little bit (say 10 minutes)" a few difficulties come to mind. To represent F_0 through F_{20} requires the computation of

$$F_i = \sum a_{ilmn}\mu^l\sigma^m\epsilon^n \quad 0\leqslant i\leqslant 20,\ \ 0\leqslant l\leqslant 10,\ \ 0\leqslant m\leqslant 18,\ \ 0\leqslant n\leqslant 9.$$

Using the most straightforward representation, some 43000 coefficients must be computed and perhaps stored. A similar sized array might be convenient for the G series. Of course most of these entries are zeros, and need never be examined, but then some of them are sufficiently large as to require double-precision arithmetic on the CDC 6400 (or similar machine). The programming to get around these problems makes the program less trivial, and as for the grocery store computer, probably makes it a less than likely prospect.

Let us assume we have a sufficiently large word size to store all coefficients of interest. The computation through F_{20} and G_{20} seems to take, in the naive approach, some 12 multiplications and 11 additions, plus 8 subscript calculations of 4-dimensional arrays (65 operations all told) for each of the 70,000 or so coefficients. If each such operation takes 1 microsecond, we are in the ballpark of 4.5 seconds. If SCHOONSCHIP takes 8.5 seconds at \$.20 per second, we have saved about \$.80, assuming our time debugging, keypunching, etc. is free, and our program works the first time through.

To test out our thesis a bit more thoroughly, we assigned a graduate student the task of computing the F and G series on the CDC6400 at Berkeley, using Fortran and nothing more. We hoped for a reasonably clever program, for example one which would not store all the results, but print out as it went along. These were compared to some "canned" solutions in MACSYMA. Using the measure of CPU time *alone*, the results were:

FORTRAN on CDC 6400	12.5 sec
MACSYMA rational function representation	3.5 sec*
MACSYMA general representation	22.2 sec*

Thus measured by CPU time alone, MACSYMA and FORTRAN perform at comparable speeds. But this measure is misleading because it doesn't take into account the expense of programming and debugging the FORTRAN program, not only in CPU time but in programmer time as well (between ten and twenty hours for the latter). The realization of the F and G series as a recurrence relation among coefficients is indeed mathematically trivial, as is the implementation of the recurrence relation as a FORTRAN program. However, in the real world where all of us have limited time, is it worth the valuable hours of a scientist's time to make such a laborious conversion when a symbolic manipulation language allows the computer to do the drudge work? Given the limitations of FORTRAN as a programming language, the task is made even more laborious than necessary: the language forbids the use of zero subscripts in arrays making the natural implementation of the recurrence relation impossible to implement directly. Such a problem could be solved by using a better-designed high level language, but the programmer will still run up against word-size limitations in integer representation. Even double precision floating point numbers on the CDC 6400 cannot retain complete accuracy beyond F_{24}. Moreover, the very "trivial" nature of the computation lends itself to errors, as is evidenced by the published formulation of the recurrence relation, which contains subscript errors in several places. The correct formulation is:

$$\begin{aligned} a_{i+1,l,m,n} &= -(3l+2(m-1)+2n)\,a_{i,l,m-1,n}+(m+1)\,a_{i,l,m+1,n-1} \\ &\quad -(n+1)\,a_{i,l-1,m-1,n+1}-b_{i,l-1,m,n} \\ b_{i+1,l,m,n} &= a_{i,l,m,n}-(3l+2(m-1)+2n)\,b_{i,l,m-1,n} \\ &\quad +(m+1)\,b_{i,l,m+1,n-1}-(n+1)\,b_{i,l-1,m-1,n+1} \end{aligned}$$

The student who implemented the Fortran program was able to save time and space by knowing the maximum subscript of nonzero coefficients in each dimension of the array of coefficients, for each F_i and G_i. This information was obtained by noting patterns appearing in a visual scan of the MACSYMA output: it is not apparent from the recurrence relation.

Even if one is willing to sacrifice some accuracy in the coefficients, and even if not storing the earlier terms in the series, it is impossible to compute beyond F_{25} within the memory of the CDC 6400.

* including garbage collection time on a DEC KL-10 computer.

Thus the example chosen to illustrate the superiority of tailor-made solutions over general symbolic manipulation languages fails to make its point. In the example of the F and G Series, it is the tailor-made solution that runs into limitations more quickly, as well as consuming much more of a programmer's time.

Sincerely yours,

Richard J. Fateman
Associate Professor, Computer Science

Attachments: program listings

The Editor notes that Dr. Veltman's article entitled, "Algebraic Techniques", is published in Computer Physics Communications (3 supplement, (1972), pp. 75-78). The Bulletin welcomes different viewpoints on this subject.