Computer Algebra and the three 'E's: Efficiency, Elegance and Expressiveness

James H. Davenport & John Fitch
Department of Computer Science
University of Bath, Bath BA2 7AY
United Kingdom
{J.H.Davenport,J.P.Fitch}@bath.ac.uk

June 18, 2007

1 Introduction

What author of a programming language would not claim that the 3 'E's were the goals? Nevertheless, we claim that computer algebra does lead to particular emphases, and constraints, in these areas.

We restrict "efficiency" to mean machine efficiency, since the other 'E's cover programmer efficiency. For the sake of clarity, we describe as "expressiveness", what can be expressed in the language, and "elegance" as how it can be expressed.

2 Efficiency

Most programming languages claim efficiency, even when their authors are dead[1]. Times have moved on, but there is still a requirement for efficiency in terms of time and space in computer algebra.

Large data structures and the need for efficiency lead to techniques such as only working modulo word-sized primes, or packing exponents several to a word. These techniques may, and generally do, lead to high-performance, but require specialised, software components, such as polynomials modulo a small prime[2], which do not generalise to cases beyond those envisaged by the designers [6].

Another illustration of the difficulties of genericity comes when we consider Gaussian elimination in sparse matrices. Here we clearly want to use a variant of Dodgson/Bareiss [2, 7] to avoid intermediate expression swell, but also some

[1] "Fortran was also extremely efficient, running as fast as programs painstakingly hand-coded by the programming elite, who worked in arcane machine languages. This was a feat considered impossible before Fortran." [20]

[2] Before these were available in Maple, non-standard techniques [3] were required to get performance.

sparsity heuristics. But the coefficients can range from large expressions to integers, most of them very small, where efficiency of storage and operation dispatch are critical [13], to the point where we would like to store (most) integers in a single byte, and avoid any dispatch overhead altogether.

Axiom [14] supported a concept of "special-case compilation", where the same code could be compiled generically and for special values of the (type) parameters, leading to in-lining etc. This or some equivalent technique is needed to bridge the genericity/efficiency gap. Furthermore, it must be (essentially) automatic, otherwise one is liable to see the code bloat that apparently bedevilled Reduce 4 [12].

However, we must admit that automated support can only go so far in providing efficiency: at some point one has to "get one's hands dirty". For example, the variety of exponent representations supported automatically by Singular [17] can only be provided by a fairly intricate piece of C/machine code. The trick then is, as Singular does, to hide this from the user. We should note that Singular is 'sound', in the sense that the choice Singular makes is in terms of the number of variables in the input, which fixes the number of variables throughout the calculation. Similarly, it would be possible (though the authors do not know of any instance) to imagine a Gröbner-oriented engine which chose a representation based on the input total degree, and (globally) changed representation if this increased, which can only happen at fixed points in Buchberger's algorithm.

3 Elegance

Compare the following.

$$\frac{-b+\sqrt{b^2-4ac}}{2a} \tag{1}$$

```
\frac{-b+\sqrt{b^2-4ac}}{2a}

(-b+SQRT(b^2-4*a*c))/(2*a)

(/ (+ (- b) (SQRT (- (^ b 2) (* 4 a c)))) (* 2 a))

(divide (plus (minus b) (sqrt (minus (power b 2) (times 4 a c))))
        (times 2 a))
```

Of course, equation (1) is what we would like to see. Given the developments in mathematical editing [1, 8, 19, and many others], we could reasonably expect the front end to mathematical languages of the future to present (and probably edit) in the format of (1), so we would argue that the problem of elegance *of appearance* has largely been moved elsewhere, while elegance of functionality is effectively expressiveness.

If this is true of *input* expressiveness, it is certainly not true of *output* expressiveness, which is related to the looser meanings of "simplication" [15]. Despite

numerous attempts [1, 11, for example], there are still large gaps between what we see and what we would like to see.

Part of the problem is that input beauty is "in the eyes of the writer", an envisionable character, whereas output beauty is "in the eyes of the (unknown) beholder" — some early examples of the challenges this poses are in [10]. It could certainly be argued that this is an area where computer algebra ought to re-discover its roots in artificial intelligence.

4 Expressiveness

However, equation (1) is not what we see in most textbooks: we see

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}. \tag{2}$$

This in itself does not look too dangerous, we merely need to widen the addition and division operators to be set-valued, as for example in SETL [18]. This looks like a road paved with a good intention, but, at least naïvely, it can lead to hell *in this case* (it *is* a useful facility in many other circumstances, such as dividing a vector by its norm).

The generally-given solution to the cubic $x^3 + bc + c$,

$$\frac{1}{6}\sqrt[3]{-108\,c + 12\sqrt{12\,b^3 + 81\,c^2}} - \frac{2b}{\sqrt[3]{-108\,c + 12\sqrt{12\,b^3 + 81\,c^2}}}, \tag{3}$$

is not three-valued at all [16], and this is a manifestation of the general problem of associating the "right" choices among multi-valued expressions. A computer scientist might feel tempted to 'solve' the problem by writing equation 3 as

$$\left(\lambda x.\frac{1}{6}x - \frac{2b}{x}\right)\sqrt[3]{-108\,c + 12\sqrt{12\,b^3 + 81\,c^2}}, \tag{4}$$

but this still appears to be six-valued (as indeed any similar expression

$$\left(\lambda x.\frac{1}{6}x - \frac{2b}{x}\right)\sqrt[3]{A + \sqrt{B}}$$

would be).

A similar problem is seen with interval arithmetic, where to get the "correct" results we need to know the dependencies between objects. A simple example (see [5] for more) is that of $x(1 - y)$ when $x, y \in [0, 1]$. This is also in $[0, 1]$ but its specialisation $x(1 - x) \in [0, 0.25]$.

For these reasons and others, extensions to computer algebra systems to support multi-valued objects have generally been of limited appeal. [4, 9]

A different problem, only partly within the scope of this workshop, concerns mathematical expressiveness. If a variable x appears, most systems have an in-built prejudice as to its range of values. This is rarely documented but seems to be as in table 1. Of these, only Maple, with its `assume` facility [21], lets the user change this prejudice.

Table 1: Systems and their views of Domains

System	Reduce	Macsyma	Maple	Mathematica
Domain	$\mathbf{R}^{\geq 0}$	$\mathbf{R}$	$\mathbf{C}$	$\mathbf{C}$

5 Conclusions

Computer algebra has made significant advances in the 3 E's since its early days in the 1960s. We have moved from simple hope of getting an answer to wishing to make it easy to use, and hence Elegance, useful in its results (Expressiveness) and capable of solving the large problems which still requires Efficiency, some of which can be obtained by a suitably expressive programming language, and some of which still requires hard work.

References

[1] O. Arsac, S. Dalmas, and M. Gaëtano. The Design of a Customizable Component to Display and Edit Formulae. In S. Dooley, editor, *Proceedings ISSAC '99*, pages 283–290, 1999.

[2] E.H. Bareiss. Sylvester's Identity and Multistep Integer-preserving Gaussian Elimination. *Math. Comp.*, 22:565–578, 1968.

[3] B.W. Char, K.O. Geddes, and G.H. Gonnet. GCDHEU: Heuristic Polynomial GCD Algorithm Based on Integer GCD Computation. In J.P. Fitch, editor, *Proceedings EUROSAM 84*, pages 285–296, 1984.

[4] J.H. Davenport and C.R. Faure. The "Unknown" in Computer Algebra. *Programmirovanie (Jan. 1994)*, pages 4–10, 1994.

[5] J.H. Davenport and H.-C. Fischer. Manipulation of Expressions. *Improving Floating-Point Programming*, pages 149–167, 1990.

[6] J.H. Davenport, J.A. Padget, and J.P. Fitch. On Symbolic Mathematical Computation. *Comm. ACM (ACM Forum)*, 28:1273–1274, 1985.

[7] C.L. Dodgson. Condensation of determinants, being a new and brief method for computing their algebraic value. *Proc. Roy. Soc. Ser. A*, 15:150–155, 1866.

[8] S.S. Dooley. Editing Mathematical Content and Presentation Markup in Interactive Mathematical Documents. In T. Mora, editor, *Proceedings ISSAC 2002*, pages 55–62, 2002.

[9] C. Faure, J.H. Davenport, and H. Naciri. Multi-valued Computer Algebra. Technical Report 4001, INRIA, 2000.

[10] R. Fenichel. An On-line System for Algebraic Manipulation. Technical Report MAC-TR-35, M.I.T., 1966.

[11] A.C. Hearn. Structure: the Key to Improved Algebraic Computation. In N. Inada and T. Soma, editors, *Proceedings 2nd. RIKEN Symp. Symbolic and Algebraic Computation*, pages 215–230, 1985.

[12] A.C. Hearn and E. Schrüfer. A Computer Algebra System based on Order-sorted Algebra. *J. Symbolic Comp.*, 19:65–77, 1995.

[13] A.J. Holt and J.H. Davenport. Resolving Large Prime(s) Variants for Discrete Logarithm Computation. In P.G. Farrell, editor, *Proceedings 9th IMA Conf. Coding and Cryptography*, pages 207–222, 2003.

[14] R.D. Jenks and R.S. Sutor. AXIOM: The Scientific Computation System. *Springer-Verlag*, 1992.

[15] J. Moses. Algebraic Simplification - A Guide for the Perplexed. *Comm. ACM*, 14:527–537, 1971.

[16] R.W.D. Nickalls. A new approach to solving the cubic: Cardan's solution revealed. *Math. Gazette*, 77:354–359, 1993.

[17] H. Schönemann. Singular in a Framework for Polynomial Computations. *Algebra*, pages 163–176, 2003.

[18] J.T. Schwartz, R.B.K. Dewar, E. Dubinsky, and E. Schonberg. Programming with Sets: An Introduction to SETL. *Springer-Verlag*, 1986.

[19] E. Smirnova and S.M. Watt. Notation Selection in Mathematical Computing Environments. In *Proceedings Transgressive Computing 2006*, pages 339–355, 2006.

[20] New York Times. Computing Pioneer Backus Dies. `http://www.nytimes.com/2007/03/20/business/20backus.html?_r=1&adxnnl=1&oref=slogin&adxnnlx=1174472570-nB7vbQKJb5SeT53vVfnlSg`, 2007.

[21] T. Weibel and G.H. Gonnet. An Assume Facility for CAS with a Sample Implementation for Maple. In *Proceedings DISCO '92*, 1993.