A CLOSED COMPUTER ARITHMETIC

F. W. J. OLVER

Institute for Physical Science and Technology, University of Maryland,
College Park, Maryland 20742, U.S.A.

ABSTRACT

Two closely related new systems of computer arithmetic are proposed. It is shown that both are closed under arithmetic operations in finite-precision arithmetic, thereby offering a permanent solution to the problems of overflow and underflow. Other advantages of the new systems pertaining to precision are described, and there is also a brief discussion of possible ways of hardware implementation.

1. Introduction and Summary

The floating-point (fℓp) system was adopted almost universally for computer arithmetic soon after the introduction of electronic computers in the early 1950's. Since then it has served the computing community remarkably well. Nevertheless there are persistent problems associated with the system. If this were not the case, then it would not have taken a large and distinguished committee many years to formulate the recent IEEE standard;[11] indeed, there may have been no need for such a standard.

Foremost among the problems associated with the fℓp system is failure caused by overflow or underflow. Anyone who has attempted to produce robust software is painfully aware of these problems; see, for example, references 1, 5. The reason overflow and underflow occur, of course, is that the fℓp system is not closed: starting from any two representable numbers it is always possible to generate numbers that lie outside the representable set by a finite number of additions, subtractions, multiplications or divisions, excluding division by zero. All modifications of the fℓp system, including for example those developed by Hull and his co-workers[9] and by Matsui and Iri,[14] also inevitably suffer from lack of closure.

Does a closed system of arithmetic exist? If it does, then we would expect it to be more complicated and costlier to execute than the fℓp system. As a result, the question has remained largely of academic interest. However, phenomenal advances in computer technology suggest that implementation of a closed system in an economic manner might now be feasible. Furthermore, although execution speed of arithmetic operations is always bound to be slower, some of the loss may be recoverable by the use of simpler algorithms. Human effort, too, would be saved by simplifications in the construction and debugging of programs.

In §2 we describe a system of computer arithmetic, called *level-index* (ℓi) arithmetic, together with a modification called *symmetric level-index* (sℓi) arithmetic. This section also outlines algorithms for implementing arithmetic operations in these systems.

In §3 we prove that the ℓi and sℓi systems are closed.

In §4 we discuss various aspects of the precision of the new systems, including some comparisons with the precision of the fℓp system. §5 supplies an illustrative example. In §6 we discuss briefly hardware implementation, and in the final section, §7, we present our main conclusions.

For fuller treatments of the topics in this paper, the reader may consult references 2, 3 in the case of §2, reference 13 in the case of §§3-5, and references 16, 19 in the case of §6.

2. The ℓi and sℓi Systems

The ℓi system is based on the mapping

$$X = \pm\phi(x), \tag{2.1}$$

and its inverse

$$x = \psi(|X|). \tag{2.2}$$

Here X is any real number, and its image (or "ℓi image") x is a nonnegative number that is stored internally in the computer (along with the sign in (2.1)) to represent X. The functions ϕ and ψ are "generalized" exponentials and logarithms, defined by

$$\phi(x) = x \quad (0 \le x < 1), \tag{2.3}$$

$$\phi(x) = e^{\phi(x-1)} \quad (x \ge 1), \tag{2.4}$$

$$\psi(X) = X \quad (0 \le X < 1), \tag{2.5}$$

$$\psi(X) = \psi(\ell n\, X) + 1 \quad (X \ge 1). \tag{2.6}$$

Both $\phi(x)$ and $\psi(X)$ are increasing functions, with continuous first derivatives when $0 \le x < \infty$ and $0 \le X < \infty$.

The image x is stored in fixed absolute precision. Its integer part ℓ and fractional part f, say, are respectively the *level* and *index* of X. The reason for these names is that in consequence of (2.1), (2.3) and (2.4) X can be expressed explicitly in the form

$$X = \pm e^{e^{\cdot^{\cdot^{e^{f}}}}},$$

where the exponentiations are carried out ℓ times. (When $\ell = 0$, we have $X = \pm f$.)

Suppose that we wish to subtract two numbers in the ℓi system. That is, we are given their images x and y, with $x \geq y \geq 0$, and we need to find z to satisfy the equation

$$\phi(z) = \phi(x) - \phi(y).$$

Let the result of decomposing x, y and z into their integer and fractional parts be denoted by

$$x = \ell + f, \quad y = m + g, \quad z = n + h.$$

If $\ell = m = 0$, then the calculation of n and h is trivial: $n = 0$ and $h = f - g$. If $\ell > 0$, then we compute three sequences $\{a_j\}$, $\{b_j\}$ and $\{c_j\}$, the members of which are defined by

$$a_j = 1/\phi(x-j), \quad b_j = a_j\phi(y-j), \quad c_j = a_j\phi(z-j).$$

The a_j are required for $j = \ell - 1, \ell - 2, \ldots, 0$. The b_j are required for $j = m - 1, m - 2, \ldots, 0$, except that we need b_0 also when $m = 0$. The sequence c_j, $j = 0,1,\ldots$ is terminated when $c_j < a_j$ or $j = \ell - 1$, whichever comes first.

Recurrence relations and initial values are supplied by

$$a_{j-1} = e^{-1/a_j}, \quad a_{\ell-1} = e^{-f},$$

$$b_{j-1} = e^{-(1-b_j)/a_j},$$

$$b_{m-1} = a_{m-1}e^{g} \ (m \geq 1), \quad b_0 = a_0 g \ (m = 0),$$

$$c_j = 1 + a_j \ell n \, c_{j-1}, \quad c_0 = 1 - b_0.$$

If $c_j (0 \leq j \leq \ell-1)$ is the first member of its sequence to satisfy $c_j < a_j$, then

$$n = j, \quad h = c_j/a_j,$$

However, if $c_j \geq a_j$ for $j = 0,1,\ldots,\ell - 1$, then

$$n = \ell, \quad h = f + \ell n \, c_{\ell-1}.$$

These relations are derivable from the definitions (2.1) to (2.6). It should be noted, incidentally, that the recurrence relations for the b_j and c_j are the same, but they are applied in opposite directions. Each of the quantities a_j, b_j and c_j lies in the interval $[0,1]$, and in reference 3 an error analysis is given to show that the algorithm can be executed in fixed-point arithmetic.

The algorithm for addition is similar to that for subtraction; the main change is to replace the equation $c_0 = 1 - b_0$ by $c_0 = 1 + b_0$. Multiplication and division are equivalent to addition and subtraction at a lower level. For example, if

$$\phi(z) = \phi(x)\phi(y), \tag{2.7}$$

with $x \geq 1$ and $y \geq 1$, then on taking logarithms we have

$$\phi(z-1) = \phi(x-1) + \phi(y-1). \tag{2.8}$$

Further details will be found in reference 3.

At level zero the ℓi system functions as a fixed-point system; compare (2.3). Often, however, we wish to preserve the relative precision of small numbers; in consequence an alternative way of representing numbers in the interval $[0,1]$ is needed. The $s\ell i$ system achieves this by using reciprocals of ℓi numbers, just as the $f\ell p$ system attains the same end by use of negative exponents. Thus in the $s\ell i$ system we have

$$X = \pm\{\phi(x)\}^{\pm 1},$$

the ambiguous sign in the exponent being positive or negative according as $|X| \gtrless 1$. Equivalently, the $s\ell i$ system can be represented by the mappings

$$X = \pm\Phi(x), \quad x = \Psi(|X|),$$

in which

$$\Phi(x) = 1/\phi(1-x) \quad (x < 0),$$

$$\Phi(x) = \phi(1+x) \quad (x \geq 0),$$

$$\Psi(X) = 1-\psi(1/X) \quad (0 < X < 1),$$

$$\Psi(X) = \psi(X) - 1 \quad (X \geq 1),$$

and $\phi(\cdot)$ and $\psi(\cdot)$ are defined as above. Again, both $\phi(\cdot)$ and $\Psi(\cdot)$ are increasing functions, with continuous first derivatives, in their intervals of definition. Algorithms for arithmetic operations in the $s\ell i$ system are obtainable by modification of those for the ℓi system; see reference 4.

3. Proof of Closure

To fix ideas, let us suppose that the internal arithmetic base of the computer is r, and the fractional parts of the ℓi images are stored to d r-nary places. Then if we add any representable number $\phi(x)$ to itself the stored ℓi image of the sum will not exceed x, provided that

$$\phi(x+cr^{-d}) > 2\phi(x). \tag{3.1}$$

Here c is a positive constant: if the ℓi arithmetic processor computed to infinite precision, then we would have $c = 1$ for chopping,

and $c = \frac{1}{2}$ for rounding. By using a sufficient number of guard digits in the processor we can approach these values arbitrarily closely, but the precise value of c is not really important. On taking logarithms the inequality (3.1) becomes

$$\phi(x-1+cr^{-d}) > \phi(x-1) + \ell n\ 2.$$

If $x > 2$, then $\phi'(x-1)$ is increasing. Hence by application of the mean-value theorem we see that the last inequality is satisfied when

$$cr^{-d}\phi'(x-1) \geq \ell n\ 2.$$

This is equivalent to

$$x \geq \theta(c^{-1}r^{d}\ell n\ 2),$$

where $\theta(t)$ denotes the root of the equation

$$\phi'(x-1) = t.$$

Sample values of $\theta(t)$, computed with the aid of Newton's rule, are as follows:

$$\theta(2^{32}) = 5.06..., \quad \theta(2^{64}) = 5.26...,$$

$$\theta(2^{1024}) = 5.63..., \quad \theta(2^{5,502,869}) = 6.00... .$$

Consider now the set A of all numbers generated by the addition or subtraction of any two numbers, beginning with any pair of numbers whose ℓi images are less than $\theta(c^{-1}r^{d}\ell n\ 2)$. Assume that the ℓi images of the members of A are generated by arithmetic operations in the ℓi system and stored to d r-nary places. In consequence of the result just proved, the absolute values of the members of A are bounded by $\phi\{\theta(c^{-1}r^{d}\ell n\ 2)\}$. If we now extend the arithmetic processes to include multiplication and division, other than division by zero, then the set of numbers so obtained is bounded in absolute value by $\phi\{\theta(c^{-1}r^{d}\ell n\ 2) + 1\}$. This is a consequence of the equivalence of multiplication and division to addition and subtraction at one level below; compare (2.7) and (2.8). See also reference 13.

We have therefore shown that when the ℓi system is used with any finite-precision arithmetic, there is a subset of its set of representable numbers that is closed under the operations of addition, subtraction, multiplication and division, excluding division by zero. Obviously, the same conclusion also applies to the $s\ell i$ system.

With the aid of the numerical values of $\theta(t)$ quoted above, we see that when $r = 2$ and $d = 32$, that is, with 32 bits assigned to the storage of the fractional part of the ℓi image, an upper bound for numbers generated by addition and subtraction is $\phi(5.07)$, whether the abbreviation mode be chopping or rounding. For multiplication and division the corresponding bound is $\phi(6.07)$. To raise the overall upper bound from $\phi(6.07)$ to $\phi(7)$, or more, we would need a wordlength in excess of 5,500,000 bits. Thus in practice levels beyond 6 will not be entered; in consequence *it will always suffice to allocate 3 bits to the storage of the integer part of the* ℓi *image.*

4. Precision

The main advantage of the ℓi and $s\ell i$ systems over other systems is the closure property established in the preceding section. However, there are also advantages pertaining to precision.

In the first place, because the mapping functions $\phi(x)$, $\psi(X)$, $\Phi(x)$ and $\Psi(X)$ have continuous derivatives, the systems are free from the annoying phenomenon of wobbling precision.[6]

Next, there is no need to introduce gradual underflow,[7,8] with its injurious effects on error analysis. In a sense, gradual underflow is already incorporated in a natural manner in the ℓi system at level zero, and in the $s\ell i$ system underflow is absent because of the closure property.

As for error analysis, the ℓi and $s\ell i$ systems have their own error measure, namely the absolute error in the ℓi and $s\ell i$ images. This measure is called *generalized precision*,[2] since it reduces to absolute error at level zero (in ℓi) and to relative error, or more precisely relative precision,[15] at level one (in ℓi). Moreover, in contrast to relative error, generalized precision has the elegant and useful property of being a metric.

Lastly, in order to compare directly local precisions* of the $f\ell p$ and ℓi (or $s\ell i$) systems for a given wordlength, we need to convert from generalized precision to relative precision. This comparison is carried out in some detail in reference 13, and we merely sketch the main results in the case of the ℓi system.

For a given mantissa length the relative precision of the $f\ell p$ system is constant, within a factor r (the internal base). For a given index length the relative precision of the ℓi system varies. Its measure is least for numbers $X = \phi(x)$ at level one, and increases steadily as X enters higher levels. At $X = 1$, the relative precision of the ℓi system is better than that of the IEEE $f\ell p$ system[11] by a factor of 32 in single precision and 256 in double precision. The relative precisions become approximately the same in the ranges $2^{11} < X < 2^{18}$ (single precision) and $2^{44} < X < 2^{70}$ (double precision). At higher values of X the $f\ell p$ system gains, up to a factor of about 37 just before overflow in single precision ($X = 2^{127}$), and of about 68 just before overflow in double precision ($X = 2^{1023}$). In consequence of these variations the comparative accuracy of results computed in the two systems will depend on the magnitude of these results as well as on the magnitudes of the numbers that appear during the intermediate steps. Unless most of these numbers happen to be in the upper part of the representable $f\ell p$ ranges (in which case the danger of overflow is increased),

*By "local precision" we mean the distance between consecutive representable numbers.

the ℓi system can be expected to yield the higher accuracy.

After X increases beyond the overflow limits, the fℓp system fails. The ℓi system still functions and the local relative precision continues to increase, becoming of order unity at $X = \phi\{\theta(c^{-1}r^{d}\ell n\ 2)\}$ that is, at "infinity" for the ℓi processes of addition and subtraction (§3). It might be argued that this gradual erosion of relative precision renders the ℓi system ineffective for very large numbers. However, relative precision is the appropriate error measure for the fℓp system, not the ℓi system. One might as well argue that the erosion of absolute precision that accompanied the change from the fixed-point system to the fℓp system would have rendered the fℓp system ineffective for large numbers.

5. Example

A good example of the power of the new arithmetics is afforded by the computation of the binomial probability distribution I(n,k,p), defined by

$$I(n,k,p) = \sum_{j=0}^{k} \binom{n}{j} p^{j}(1-p)^{n-j},$$

with $0 \le k \le n$ and $0 < p < 1$. Over six pages of Sterbenz's well-known book[18] are devoted to this problem. Sterbenz discusses failures caused by overflow and underflow in fℓp arithmetic, and provides an algorithm to overcome these difficulties, based on careful rescaling. The algorithm is for individual terms in the sum, and is quite complicated. In contrast, a simple recursive algorithm based on the relations $y_0 = (1-p)^n$,

$$y_j = \frac{n-j+1}{j}\,\frac{p}{1-p}\,y_{j-1} \qquad (j = 1,2,\ldots k),$$

and

$$I(n,k,p) = \sum_{j=0}^{k} y_j,$$

can be executed straightforwardly in sℓi arithmetic and yields results of comparable accuracy to those generated by Sterbenz's algorithm.

The advantages of the sℓi algorithm are first that the construction and debugging of the program are less time-consuming. Secondly, the sℓi algorithm entails far fewer arithmetic operations, which offsets some of the increase in execution time needed for sℓi arithmetic.

Further details of this example, including computed results, will be found in reference 13.

6. Hardware Implementation

So far, implementations of the ℓi and sℓi systems have been made only in software. These implementations have been somewhat cumbersome because for convenience the steps of the arithmetic algorithms (§2) have been simulated in fℓp arithmetic, rather than the more natural fixed-point arithmetic.

The next stage will be the development of fast hardware processors for evaluating the requisite exponentials and logarithms. Given numbers a, b and c in fixed-point form we need to compute e^{-a}, $e^{-1/a}$, $e^{-(1-b)/a}$ and $\ell n\ c$, also in fixed-point form. On microcomputers algorithms of the CORDIC type (originated by J. E. Volder) might be useful; see, for example, references 17, 20. For mainframe computers, however, digital parallel algorithms of the kind used for fixed-point and fℓp arithmetic operations are likely to be more effective; see, for example, reference 10, §3.2, and reference 21, Chapter 3. Considerable progress on these lines has already been made by Johnsson and Krishnaswamy,[12] and Turner and Olver.[16,19]

The ultimate objective is to have the ℓi and sℓi systems encoded on a single silicon chip. Then at the outset of a program, programmers would be able to declare whether they wish the arithmetic to be executed in fℓp, ℓi, sℓi or some other arithmetic, e.g. that of reference 9, or even fixed-point arithmetic. This declaration would be analogous to those now made in FORTRAN pertaining to single or double precision, and real or complex variables.

Of course, it is unlikely that ℓi and sℓi arithmetic operations will ever be as fast as the corresponding operations in fℓp arithmetic. However, if the ratio of the speeds could be brought to within 10, or so, then the overall execution time of a program in ℓi or sℓi arithmetic might be only two or three times that of the same program in fℓp arithmetic. This is because only a part of program execution is spent on arithmetic operations, as a rule. Moreover, as indicated in §5, freedom from overflow and underflow means that it will be possible to implement simpler algorithms in the ℓi and sℓi systems than in the fℓp system. This advantage will help to offset further some of the speed loss of the arithmetic operations. It will also lighten human effort in constructing, testing and debugging programs and software.

7. Conclusions

We have described a new form of computer arithmetic, called the level-index system, together with a modification thereof, called the symmetric level-index system. In contrast to existing systems, the ℓi and sℓi systems are closed under arithmetic operations in finite-precision arithmetic. In consequence, these systems offer a permanent solution to the problems of overflow and underflow. There are also certain other advantages pertaining to precision. Execution speeds of arithmetic operations will be slower than for the floating-point system, but this loss may be offset to some extent by the ability to use simpler programs.

We have not entered into any discussion of error analysis in the new arithmetics. This subject is investigated in reference 13.

We can summarize briefly by stating that the advantages and challenges of the new systems are similar to those that were offered in the 1950's on the introduction of the floating-point system.

Acknowledgements

The work described in this paper is part of an ongoing project by several researchers, particularly C. W. Clenshaw and P. R. Turner at the University of Lancaster, D. W. Lozier at the National Bureau of Standards, and the author at the University of Maryland. Several helpful suggestions have been made by A. Feldstein of Arizona State University. Support has been provided, in part, by the U. S. Army Research Office, Durham under Contract DAAG-29-84-K-0022, and the National Science Foundation under Grant DMS-84-19820.

References

[1] W. S. Brown, "A simple but realistic model of floating-point computation," *ACM Trans. Math. Software*, v. 7, pp. 445-480, 1981.

[2] C. W. Clenshaw and F. W. J. Olver, "Beyond floating point," *J. Assoc. Comput. Mach.*, v. 31, pp. 319-328, 1984.

[3] C. W. Clenshaw and F. W. J. Olver, "Level-index arithmetic operations," *SIAM. J. Numer. Anal.*, 1987. [In press.]

[4] C. W. Clenshaw and P. R. Turner, "The symmetric level-index system." [Submitted for publication.]

[5] W. J. Cody, "An overview of software development for special functions." *In* Lecture Notes in Mathematics, 506, *Numerical Analysis Dundee 1975*, G. A. Watson ed., Springer-Verlag, Berlin, pp. 38-48, 1976.

[6] W. J. Cody, "Basic concepts for computational software." *In* Lecture Notes in Computer Science, 142, *Problems and Methodologies in Mathematical Software Production*, P. C. Messina and A. Murli eds., Springer-Verlag, Berlin, pp. 1-23, 1982.

[7] J. T. Coonen, "Underflow and the denormalized numbers," *Computer*, v. 14, pp. 75-87, 1981.

[8] J. Demmel, "Underflow and the reliability of numerical software," *SIAM J. Sci. Statist. Comput.*, v. 5, pp. 887-919, 1984.

[9] T. E. Hull, "Precision control, exception handling and a choice of numerical algorithms." *In* Lecture Notes in Mathematics, 912, *Numerical Analysis*, G. A. Watson ed., Springer-Verlag, Berlin, pp. 169-178, 1982.

[10] K. Hwang and F. A. Briggs, *Computer Architecture and Parallel Processing*, McGraw-Hill, New York, 1984.

[11] IEEE Standard 754, *Binary Floating-Point Arithmetic*, The Institute of Electrical and Electronics Engineers, New York, 1985.

[12] S. L. Johnsson and V. Krishnaswamy "Floating-point Cordic." [Manuscript.]

[13] D. W. Lozier and F. W. J. Olver, "Closure and precision in computer arithmetic." [Manuscript.]

[14] S. Matsui and M. Iri, "An overflow/underflow-free floating-point representation of numbers," *J. Inform. Process.*, v. 4, pp. 123-133, 1981.

[15] F. W. J. Olver, "A new approach to error arithmetic," *SIAM J. Numer. Anal.*, v. 15, pp. 368-393, 1978.

[16] F. W. J. Olver and P. R. Turner, "Implementation of level-index arithmetic using partial table look-up," these proceedings.

[17] C. W. Schelin, "Calculator function approximation," *Amer. Math. Monthly*, v. 90, pp. 317-325, 1983.

[18] P. H. Sterbenz, *Floating-Point Computation*, Prentice-Hall, Englewood Cliffs, New Jersey, 1974.

[19] P. R. Turner, "Towards a fast implementation of level-index arithmetic," *Bull. Inst. Math. Appl.*, 1987. [In press.]

[20] J. S. Walther, "A unified algorithm for elementary functions," *AFIPS Conference Proc.*, v. 38, pp. 379-385, 1971.

[21] S. Waser and M. J. Flynn, *Introduction to Arithmetic for Digital Systems Designers*, Holt, Rinehart and Winston, New York, 1982.