FLOATING-POINT ARITHMETIC AND PROGRAM CORRECTNESS PROOFS

A Thesis

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment for the Degree of

Doctor of Philosophy

by

John Erick Holm

August 1980

FLOATING-POINT ARITHMETIC AND PROGRAM CORRECTNESS PROOFS

John Erick Holm, Ph.D.
Cornell University 1980

This thesis develops tight upper and lower bounds on the relative error in various schemes for performing floating-point arithmetic, proposes axioms for characterizing the significant properties embodied by these schemes, and gives examples to illustrate how these axioms may be used to reason about the correctness of floating-point programs.

Three addition schemes are considered: (1) chopped addition, (2) addition with both pre and post-adjustment rounding, and (3) addition with pre-adjustment chopping and post-adjustment rounding. Schemes for performing both rounded and chopped multiplication and division are also considered.

Our tight bounds are consistent with the commonly held opinion that a binary base minimizes the maximum relative errors in floating-point arithmetic. Also, these bounds show that one guard digit is optimal for minimizing the maximum relative errors in chopped addition. The bounds derived for each of the addition schemes considered are as tight as possible.

One guard digit and two guard bits are shown to be sufficient to round the result of an exact addition to the nearest floating-point number. We show how this scheme can be implemented using a single post-adjustment shift, no rounding overflow, and (for certain implementations) requiring no more time than an addition that chops instead of rounds.

Two approaches are considered for axiomatizing floating-point arithmetic. In one approach, a set of floating-point numbers is associated with each floating-point expression, and the assignment statement is modeled as a nondeterministic selector of one of the members in the set. In the alternative approach, the floating-point operations are modeled in terms of two cropping functions whose significant properties are characterized by a small set of axioms. In both cases, the axioms characterizing floating-point arithmetic are used with Dijkstra's weakest pre-condition calculus to provide an axiomatic framework for reasoning about floating-point programs.

Finally, the common practice of modelling the floating-point operations by a single function that chops or rounds the result of the corresponding exact operation is shown to be invalid for many implementations of floating-point arithmetic.

Biographical Sketch

John Erick Holm was born in Yokohama, Japan on August 29, 1953. His father's army career led John to attend schools in Pennsylvania, Virginia, Maryland, Utah, Germany, and Korea, where he graduated as valedictorian of his high school class in June 1971.

He subsequently entered a Co-Op work/study program with Northwestern University and Allis-Chalmers Corp., earning a B.S. with Distinction in computer science in June 1976.

John was awarded a M.S. in computer science by Cornell University in January 1979. He is currently a member of Tau Beta Pi, Phi Eta Sigma, the Association for Computing Machinery, and the IEEE Computer Society.

To my parents

for their constant inspiration and support

Acknowledgements

I would like to express my deep gratitude to my thesis advisor, Professor David Gries, for his always ready and cheerful encouragement, advice, and support. His lessons on clarity and simplicity have made a lasting impression on me.

Also, I would like to thank Professor Frank Luk for a close reading of this thesis and for his constructive comments, Dr. Gary Levin for help with the floating-point axioms, and Mr. Robert Hood for help with the typography.

Table of Contents

List of Tables and Figures

1. Introduction

Forty years after Konrad Zuse built what is probably the first mechanical floating-point binary computer [42, p. 169], it appears that floating-point arithmetic is still not well understood. The literature is fraught with errors (we shall indicate several), the hardware is still designed by expediency†, and, consequently, the software often incorporates subtle "tricks" to overcome the inadequacies of the hardware design††.

The development of fast and accurate algorithms for performing the elementary floating-point operations (+, -, ×, /) is still an active area of basic research [7,20,23,27,43,63,74], and a general theory of floating-point arithmetic is just now underway [21,41,45,46]. In fact, specification of a standard system for floating-point arithmetic is a current, hotly debated topic [3,9,33,49,59,62].

Given the state of tumult described above, the objective of this thesis is to enhance the confidence one has in reasoning about computer programs that use floating-point arithmetic. To attain this objective, we first study floating-point arithmetic to determine the effect of the number representation, various numbers of guard digits, and various rounding or chopping schemes upon the accuracy of the arithmetic. Then we present two models of floating-point arithmetic and show how each lends itself to

---

†Viz. any computer that doesn't perform rounded arithmetic (IBM 360/370 [35,36] and Univac 1100 [41]), or that rounds incorrectly (CDC 3000, 6000, and Cyber-70 [26,41]).

††Viz. Kahan's fudge-factor of 0.46 in his algorithm for computing the sum of a list of numbers [39]. See also Dekker [11] and Liddiard [49].

rigorous correctness proofs of the Floyd-Hoare style. The motivation for developing proofs in this style is given in section 3.2.

2. Floating-Point Hardware Design Considerations

> ...the rounding strategy inherent to a floating-point microprogram of a computer (or calculator) is the worst place for a machine designer to demonstrate originality, in particular if his own experience with numerical calculation is rather limited. The chaos of careless and exotic rounding strategies in our present computers seriously impedes the production of clean numerical software. Dodges and tricks, costly to develop, which might be necessary to overcome the difficulties with rounding effects on one computer may be unnecessary or even damaging on another....
>
> -- Christian Reinsch [59]

2.1 Overview

In section 2, we derive tight upper and lower bounds on the relative errors in floating-point arithmetic as functions of the base and the number of digits employed for representing the floating-point numbers. Using these bounds, we argue that a binary representation is optimal for minimizing the maximum relative errors. Also, it follows that unbiased rounding schemes yield <u>greater</u> worst case negative relative errors than those that round with positive bias and <u>greater</u> worst case positive relative errors than those that round with negative bias.

We investigate the hardware register requirements for performing floating-point arithmetic and show that one guard digit is optimal for minimizing the maximum relative errors in chopped addition. We show that one guard digit and two guard bits are sufficient to round the result of an exact addition to the nearest floating-point number. Finally, we show how this scheme can be implemented with a single post-adjustment shift, no rounding overflow, and requiring no more time than an addition that chops instead of rounds.

2.2 *Floating-Point Numbers and Operations*

A *floating-point number* (fp number) is a number that can be expressed in the form

$$(s) \sum_{i=1}^{t} d_i \, \beta^{p-i} \qquad (2.2.1)$$

where $\beta \geq 2$ is the *base*, p is an *exponent* in some fixed range, $-M_1 \leq p \leq M_2$ ($M_1, M_2 \geq 0$), $t \geq 1$ is the number of *digits*, each digit lies in the range $0 \leq d_i < \beta$, and $s = \pm 1$ is the *sign*. We shall assume that all fp numbers are *normalized*; hence, if some $d_i \neq 0$ then $d_1 \neq 0$.

For any real number x, let $\lfloor x \rfloor$ denote the greatest integer $\leq x$ (or floor(x)). When we say that $x \geq 0$ is *chopped to q digits*, we mean that x, expressed in the normalized form of (2.2.1) with unbounded p and t, is replaced by

$$\lfloor x \beta^{q-p} \rfloor \beta^{p-q}$$

When we say that $x \geq 0$ is *rounded to q digits*, we mean that x is replaced by

$$\lfloor x \beta^{q-p} \rfloor \beta^{p-q} + \Delta r$$

where

$$\Delta r = 0 \qquad \text{if} \quad x - \lfloor x \beta^{q-p} \rfloor \beta^{p-q} < \tfrac{1}{2} \beta^{p-q}$$

$$\Delta r = \beta^{p-q} \qquad \text{if} \quad x - \lfloor x \beta^{q-p} \rfloor \beta^{p-q} \geq \tfrac{1}{2} \beta^{p-q}$$

For $x < 0$, x *rounded* or *chopped to q digits* is defined to be the negative of $-x$ *rounded* or *chopped to q digits*, respectively. Henceforth, we use the word *cropping* to denote either rounding or chopping.

Suppose that some cropping scheme is used in performing arithmetic upon fp numbers in a t-digit *accumulator* with $g \geq 0$ [illegible] *digit registers* and a one bit *overflow register*. The guard digit registers are used to extend the precision of the accumulator to $t+g$ digits, for accumulating intermediate results generated by an fp operation. (See figure 2.2.1.)

```
 _    ______________________________    ____
|_|  |______________________________|  |____|
OB   <------------ AC ------------->    GD
```

Figure 2.2.1. Layout of registers for performing fp arithmetic. (OB, AC, and GD denote the overflow bit, accumulator, and guard digit registers, respectively.)

For the purposes of this thesis, an *fp operator*, represented by one of $\oplus$, $\ominus$, $\otimes$, $\oslash$, denotes a hardware or software implementation of the operations addition, subtraction, multiplication, and division, respectively.

Formally, let $\mathbf{F} = \mathbf{F}(\beta, t, M_1, M_2)$ denote the set of fp numbers defined by specific fixed values of β, t, M_1, and M_2. Then, for a given fp implementation, the fp operations are partial functions of the form $\mathbf{F} \times \mathbf{F} \rightarrow \mathbf{F}$.

These definitions illustrate the chief differences between exact and fp arithmetic: because of limitations in *range* (i.e. M_1 and M_2) and *precision* (i.e. t), the numbers that can be used as inputs, outputs, or intermediate operands of the fp operations form a finite subset of the (infinite) set of reals, $\mathbb{R}$. In fact, the fp operations have been characterized above as *partial* functions, because usually none of them is even defined for all members of $\mathbf{F} \times \mathbf{F}$.

If an fp operation is not defined because the normalized result of the corresponding exact operation has an exponent larger than M_2, then the exponent is said to have *overflowed*. On the other hand, if an fp operation is not defined because the normalized result of the corresponding exact operation has an exponent smaller than $-M_1$, then the exponent is said to have *underflowed*.

2.3 *Floating-Point Multiplication*

Consider $x \circledast y$, the fp product of the fp numbers x and y drawn from $\mathbb{F} = \mathbb{F}(\beta, t, M_1, M_2)$. Given that exponent overflow and underflow do not occur, we assume for any x and y in $\mathbb{F}$ that

$$x \circledast y = y \circledast x \tag{2.3.1}$$

$$-(x \circledast y) = (-x) \circledast y \tag{2.3.2}$$

$$(x = 0 \ \vee \ y = 0) \Rightarrow x \circledast y = 0 \tag{2.3.3}$$

We shall be considering the maximum relative cropping errors in $\circledast$. For this purpose, assumptions (2.3.1) - (2.3.3) allow us, without loss of generality, to restrict our attention to x and y satisfying $x, y > 0$. Also, since we shall be concerned only with relative errors, we assume without loss of generality, that the exponents (in the normalized representations) of x and y are both zero. Henceforth, the *exponent* of the fp number x refers to the exponent in the normalized representation of x.

Suppose that $x \circledast y$ is formed as follows.

(1) *Product Formation*: The product xy is determined exactly in a t digit accumulator with t guard digit registers and no overflow registers.

(2) *Post-Adjustment*: The product xy is cropped to t digits and then left-shifted 0 or more positions to place its leftmost nonzero digit in the most

significant digit position of the accumulator.

Let us define

$\Delta p \equiv$ the error incurred by discarding the least significant digits of xy ($\Delta p \geq 0$);

$\Delta r \equiv$ the amount by which the t-th digit of xy is incremented when xy is cropped to t digits.

Thus, if cropping is performed by chopping, then

$$\Delta r = 0$$

otherwise cropping is performed by rounding and

$$\Delta r = 0 \qquad \text{if} \quad \Delta p < \frac{1}{2}\beta^{-L-t} \tag{2.3.4}$$

$$\Delta r = \beta^{-L-t} \qquad \text{if} \quad \Delta p \geq \frac{1}{2}\beta^{-L-t} \tag{2.3.5}$$

where L is the number of leading zeros in xy prior to post-adjustment cropping ($L \geq 0$).

It follows that $x \bullet y$ can be described by

$$x \bullet y = xy - \Delta p + \Delta r \tag{2.3.6}$$

and the relative error in $x \bullet y$ can be described by

$$E_{x \bullet y} \equiv \frac{x \bullet y - xy}{xy} = \frac{\Delta r - \Delta p}{(x \bullet y - \Delta r) + \Delta p} \tag{2.3.7}$$

Since x and y are > 0 and normalized, we see from (2.2.1) that $\beta^{-1} \leq x, y \leq 1 - \beta^{-t}$. Thus,

$$\beta^{-2} \leq xy \leq 1 - 2\beta^{-t} + \beta^{-2t}$$

Consequently, $0 \leq L \leq 1$. Furthermore, if xy is rounded as described above, the result would still be < 1, and no overflow registers are needed. If a "carry" could propagate beyond the most-significant digit position in the accumulator, this phenomenon would be called rounding overflow.

Suppose cropping is performed by chopping. Since $x, y > 0$, $\Delta p \geq 0$, and in this case, $\Delta r = 0$; we see from (2.3.7) that $E_{x \circledast y} \leq 0$. That 0 is a least upper bound on $E_{x \circledast y}$ is determined by observing $E_{x \circledast y} = 0$ when $x \circledast y = xy$.

To obtain a lower bound on $E_{x \circledast y}$, observe that for $z > 0$ and $w \geq 0$, the function $f(w,z) \equiv w/(z+w)$ is maximized by minimizing z and maximizing w. Consequently, a lower bound on $E_{x \circledast y}$ is obtained by substituting in (2.3.7) the smallest possible value for $x \circledast y$ and the largest possible value for Δp.

Recall that L is the number of leading zeros in xy prior to post-adjustment cropping. For each L, the smallest possible value of $x \circledast y$ is β^{-L-1}. Also, the largest possible value of Δp is obtained when each of the digits in $xy - x \circledast y$ equals $\beta - 1$. Consequently, for each L, Δp can be no larger than $\beta^{-L}(\beta^{-t} - \beta^{-2t})$.

Summarizing for chopped $x \circledast y$,

$$
\begin{aligned}
E_{x \circledast y} &= \frac{-\Delta p}{x \circledast y + \Delta p} \geq \frac{-\beta^{-L}(\beta^{-t} - \beta^{-2t})}{\beta^{-L-1} + \beta^{-L}(\beta^{-t} - \beta^{-2t})} \\
&= \frac{-(1 - \beta^{-t})}{\beta^{t-1} + (1 - \beta^{-t})} \qquad (2.3.8) \\
&> -(1 - \beta^{-t})\,\beta^{1-t}
\end{aligned}
$$

One can verify that bound (2.3.8) is indeed attained for $\beta = 2$ and $t = 4$, with

$$x = 2^{-1} + 2^{-2} + 2^{-4} \quad \text{and} \quad y = 2^{-1} + 2^{-3} + 2^{-4}$$

Consequently, (2.3.8) is a lower bound for all $\beta \geq 2$, $t \geq 1$, and a greatest lower bound for $\beta = 2$ and $t = 4$.

Suppose now that cropping is performed by rounding. From (2.3.4) and (2.3.5) we see that the nonzero error in $x \circledast y$ takes two cases:

$$\text{Case 1.} \quad \Delta r = 0 \quad \text{and} \quad \Delta p < \tfrac{1}{2}\beta^{-L-t} \tag{2.3.9}$$

$$\text{Case 2.} \quad \Delta r = \beta^{-L-t} \quad \text{and} \quad \Delta p \geq \tfrac{1}{2}\beta^{-L-t} \tag{2.3.10}$$

Consider case 1. Since $x, y > 0$, $\Delta r = 0$, and $\Delta p \geq 0$, it follows from (2.3.7) that $E_{x\circledast y} \leq 0$, as in the case of chopped $x \circledast y$ described previously. We observed that a lower bound on $E_{x\circledast y}$ is obtained by substituting in (2.3.7) the smallest possible value for $x \circledast y$ and the largest possible value for Δp.

From (2.3.9) we have $\Delta p < \frac{1}{2}\beta^{-L-t}$. But, since Δp only accounts for error to the left of the last guard digit, it follows that, for each L, Δp can be no larger than $\beta^{-L}(\frac{1}{2}\beta^{-t} - \beta^{-2t})$. Also, as in the case of chopped $x \circledast y$, for any L the smallest value of $x \circledast y$ is β^{-L-1}.

Summarizing for case 1, we have

$$E_{x\circledast y} = \frac{-\Delta p}{x\circledast y + \Delta p} \geq \frac{-\beta^{-L}(\frac{1}{2}\beta^{-t} - \beta^{-2t})}{\beta^{-L-1} + \beta^{-L}(\frac{1}{2}\beta^{-t} - \beta^{-2t})}$$

$$= \frac{-(\frac{1}{2} - \beta^{-t})}{\beta^{t-1} + (\frac{1}{2} - \beta^{-t})} \tag{2.3.11}$$

$$> -(\tfrac{1}{2} - \beta^{-t})\,\beta^{1-t}$$

One can verify that bound (2.3.11) is indeed attained for $\beta = 2$ and any $t \geq 1$, with

$$x = y = 2^{-1} + 2^{-t}$$

Consider case 2. Since $x, y > 0$, and for any L, $\Delta r > \Delta p$, it follows from (2.3.7) that $E_{x\circledast y} > 0$. Before post-adjustment rounding, for each L, the fp product must be at least β^{-L-1}. I.e. $x\circledast y - \Delta r \geq \beta^{-L-1}$. Since $\Delta p \geq \frac{1}{2}\beta^{-L-t}$, from (2.3.6) we obtain

$$xy \;=\; (x\circledast y - \Delta r) + \Delta p \;\geq\; \beta^{-L-1} + \frac{1}{2}\beta^{-L-t}$$

Summarizing for case 2, we have

$$\begin{aligned} E_{x\circledast y} &= \frac{\Delta r - \Delta p}{(x\circledast y - \Delta r) + \Delta p} \leq \frac{\frac{1}{2}\beta^{-L-t}}{\beta^{-L-1} + \frac{1}{2}\beta^{-L-t}} \\ &= \frac{\frac{1}{2}}{\beta^{t-1} + \frac{1}{2}} \qquad\qquad (2.3.12) \\ &< \frac{1}{2}\beta^{1-t} \end{aligned}$$

One can verify that bound (2.3.12) is indeed attained for $\beta = 2$ and $t = 5$, with

$$x = 2^{-1} + 2^{-2} \qquad \text{and} \qquad y = 2^{-1} + 2^{-3} + 2^{-4}$$

Therefore, for <u>rounded</u> $x\circledast y$, we have shown (using cases 1 and 2) that

$$\frac{-(\frac{1}{2} - \beta^{-t})}{\beta^{t-1} + (\frac{1}{2} - \beta^{-t})} \leq E_{x\circledast y} \leq \frac{\frac{1}{2}}{\beta^{t-1} + \frac{1}{2}} \qquad (2.3.13)$$

where the equalities are attained for certain values of β and t.

It is straightforward to show that the first t significant digits of the product of two t-digit numbers may be obtained by accumulating partial sums in a t-digit accumulator with one guard digit and no overflow bits. If an additional guard bit is also available to indicate whether the t + 2 nd digit of the exact product is $\geq \frac{1}{2}\beta$, then results identical to those generated by the 2t-digit rounding scheme described above may be obtained.

For information concerning the statistical effect of various numbers of guard digits upon the accuracy of fp multiplication, see Goodman and Feldstein [23] and Bustoz, Feldstein, and Goodman [7].

The rounding scheme described above is said to have a positive bias, because $\Delta p \geq \frac{1}{2}\beta^{-L-t}$ is rounded away from zero. If our scheme were modified to round an error of exactly $\frac{1}{2}\beta^{-L-t}$ toward or away from zero with equal probability, then the scheme would be called unbiased. In the unbiased rounding scheme, the bound on the positive relative error would be the same as in the scheme with positive bias. However, the magnitude of the negative relative error bound would increase to the same value as the positive relative error bound. Also, the 1 guard digit + 1 guard bit rounding scheme mentioned above would have to be augmented by an additional (so called) "sticky" bit for distinguishing $\Delta p_1 = (\frac{1}{2}\beta)\beta^{-L-t-1}$ and Δp_2 satisfying

$$(\tfrac{1}{2}\beta)\beta^{-L-t-1} < \Delta p_2 < (\tfrac{1}{2}\beta + 1)\beta^{-L-t-1} \qquad (2.3.14)$$

The greater liklihood for cancellation of errors with an unbiased rounding scheme in computational procedures [43,44] is certainly adequate justification for the modicum of additional hardware cost and very slightly larger relative error bounds.

Perhaps more important than the exact value of the bounds derived in this section is the functional dependence of the bounds upon β and t which they demonstrate. For example, taking into account the fact that a factor of $\log\beta_2/\log\beta_1$ more bits are needed to represent a number in base β_1 than in base β_2, the bounds presented in this section all indicate that base 2 would incur the smallest maximum relative errors for any given number of bits. In sections 2.4 and 2.5 we shall see that base 2 yields the smallest bounds on the relative error in $\oslash$, $\oplus$, and $\ominus$, as well. This observation is particularly significant for $\oplus$ and $\ominus$, since the bounds we derive for $\oplus$ and $\ominus$ are as tight as possible.

2.4 Floating-Point Division

Consider $x \oslash y$, the fp quotient of the fp numbers x and y drawn from $\mathbb{F} = \mathbb{F}(\beta, t, M_1, M_2)$. Given that exponent overflow and underflow do not occur, we assume for any $y \neq 0$ and x in $\mathbb{F}$ that

$$-(x \oslash y) \;=\; (-x) \oslash y \;=\; x \oslash (-y) \tag{2.4.1}$$

$$x = 0 \;\Rightarrow\; x \oslash y = 0 \tag{2.4.2}$$

As in the case of fp multiplication, described in section 2.3, we shall be considering the maximum relative cropping errors in $\oslash$. For this purpose, assumptions (2.4.1) and (2.4.2) allow us, without loss of generality, to restrict our attention to x and y satisfying $x, y > 0$. Also, as in section 2.3, we assume without loss of generality that the exponents of x and y are both zero.

Suppose that $x \oslash y$ is formed as follows[†].

(1) Pre-Adjustment: x is placed in a t-digit accumulator with no overflow bits. It is then right-shifted 1 position, placing its least significant digit in a guard digit register. Denote the shifted x by x'. (Note: x is shifted to ensure $x' < y$.)

[†] This algorithm is similar to the division algorithms given by Wilkinson [70], Knuth [42], and Hill and Peterson [28].

(2) Quotient Formation: By performing exact subtractions in the t-digit accumulator with single guard digit, the first t significant digits of x'/y are determined exactly and stored in a $t+1$ digit quotient register.

(3) Post-Adjustment: The contents of the quotient register are left-shifted 0 or more positions to place the leftmost nonzero digit in the most significant digit position of the quotient register. This shifted quotient is then cropped to t digits, making use of the remainder -- the result of the final subtraction during quotient formation.

Let us define

$\Delta q \equiv$ the error incurred by discarding the remainder from the t-digit quotient ($\Delta q \geq 0$);

$\Delta r \equiv$ the amount by which the rightmost digit of the quotient is incremented when the quotient is cropped to t digits.

Thus, if cropping is performed by chopping, then

$$\Delta r = 0$$

otherwise cropping is performed by rounding and

$$\Delta r = 0 \qquad \text{if} \quad \Delta q < \frac{1}{2}\beta^{-L-t} \tag{2.4.3}$$

$$\Delta r = \beta^{-L-t} \qquad \text{if} \quad \Delta q \geq \frac{1}{2}\beta^{-L-t} \tag{2.4.4}$$

where L is the number of leading zeros in the quotient register, prior to post-adjustment shifting ($L \geq 0$).

It follows that $x \oslash y$ can be described by

$$x \oslash y = x/y - \Delta q + \Delta r \tag{2.4.5}$$

and the relative error in $x \oslash y$ can be described by

$$E_{x \oslash y} \equiv \frac{x \oslash y - x/y}{x/y} = \frac{\Delta r - \Delta q}{(x \oslash y - \Delta r) + \Delta q} \tag{2.4.6}$$

Since x and y are > 0 and normalized, we see from (2.2.1) that $\beta^{-1} \le x, y \le 1 - \beta^{-t}$. Thus, if $x < y$,

$$\beta^{-2} < \frac{\beta^{-2}}{1 - \beta^{-t}} \le \frac{x\beta^{-1}}{y} < x\beta^{-1} \le (1 - \beta^{-t})\beta^{-1} \tag{2.4.7}$$

from which it follows that the quotient has exactly one leading zero (L = 1) and rounding overflow cannot occur. Otherwise, $x \ge y$, yielding

$$\beta^{-1} \le \frac{x\beta^{-1}}{y} \le \frac{x\beta^{-1}}{\beta^{-1}} = x \le 1 - \beta^{-t} \tag{2.4.8}$$

from which it follows that the quotient has no leading zeros (L = 0), and, once again, rounding overflow cannot occur. That rounding overflow cannot occur by rounding the quotient of two normalized fp numbers was also noticed by Knuth [42].

Suppose cropping is performed by <u>chopping</u>. We use the same reasoning as that employed for chopped $x \circledast y$ to conclude that 0 is the <u>least</u> upper bound on $E_{x \oslash y}$. Similarly, to obtain a lower bound on $E_{x \oslash y}$, we can immediately conclude that $E_{x \oslash y}$ will be as negative as possible when $x \oslash y$ is replaced by β^{-L-1} and Δq is made as large as possible.

Let <u>rem</u> denote the remainder after t significant digits of x/y are obtained. By the definition of division, $rem < y\beta^{-L-t}$. Moreover, since y has no more than t significant digits,

$$rem \le (y - \beta^{-t})\beta^{-L-t}$$

Hence,

$$\Delta q = \frac{rem}{y} \le \frac{(y - \beta^{-t})\beta^{-L-t}}{y}$$

$$= (1 - \beta^{-t}/y)\beta^{-L-t}$$

$$\leq \; [1 - \beta^{-t}/(1 - \beta^{-t})]\,\beta^{-L-t}$$

Summarizing, we have for chopped $x \oslash y$,

$$E_{x\oslash y} \;=\; \frac{-\Delta q}{x\oslash y + \Delta q} \;\geq\; \frac{-\beta^{-L-t}[1 - \beta^{-t}/(1 - \beta^{-t})]}{\beta^{-L-1} + \beta^{-L-t}[1 - \beta^{-t}/(1 - \beta^{-t})]}$$

$$= \frac{-[1 - 1/(\beta^{t} - 1)]}{\beta^{t-1} + [1 - 1/(\beta^{t} - 1)]} \qquad (2.4.9)$$

$$> \; -\beta^{1-t}[1 - 1/(\beta^{t} - 1)]$$

One can verify that bound (2.4.9) is attained for $\beta = 2$ and $t = 2$, with

$$x = 2^{-1} \quad \text{and} \quad y = 2^{-1} + 2^{-2}$$

Consequently, (2.4.9) is a lower bound for all $\beta \geq 2$, $t \geq 1$, and a greatest lower bound for $\beta = 2$ and $t = 2$. We do not know whether this bound is also attained for other choices of β and t.

Suppose now that cropping is performed by rounding. From (2.4.3) and (2.4.4) we see that the nonzero error in $x\oslash y$ takes two cases:

$$\text{Case 1.} \quad \Delta r = 0 \quad \text{and} \quad \Delta q < \tfrac{1}{2}\beta^{-L-t} \qquad (2.4.10)$$

$$\text{Case 2.} \quad \Delta r = \beta^{-L-t} \quad \text{and} \quad \Delta q \geq \tfrac{1}{2}\beta^{-L-t} \qquad (2.4.11)$$

Consider case 1. From (2.4.6) and (2.4.10) it is evident that $E_{x\oslash y} \leq 0$. Once again, we see that (2.4.6) is made as negative as possible by substituting β^{-L-1} for $x\oslash y$, and the largest possible value for Δq.

Since $\Delta q < (\frac{1}{2}\beta)\,\beta^{-L-t-1}$, a maximum value of Δq is attained when the t+1st significant digit of the quotient is $\frac{1}{2}\beta - 1$, and the subsequent remainder is as large as possible. Namely,

$$\Delta q \;=\; \frac{rem}{y} \;\leq\; (\tfrac{1}{2}\beta - 1)\,\beta^{-L-t-1} + \frac{(y - \beta^{-t})\,\beta^{-L-t-1}}{y}$$

$$\leq \; (\tfrac{1}{2}\beta - 1)\,\beta^{-L-t-1} + \frac{(1 - 2\beta^{-t})\,\beta^{-L-t-1}}{1 - \beta^{-t}}$$

$$= \frac{1}{2}\beta^{-L-t}[1 - 2\beta^{-t-1}/(1 - \beta^{-t})]$$

Summarizing for case 1, we have

$$E_{x\oslash y} = \frac{-\Delta q}{x\oslash y + \Delta q} \geq \frac{-\frac{1}{2}\beta^{-L-t}[1 - 2\beta^{-t-1}/(1 - \beta^{-t})]}{\beta^{-L-1} + \frac{1}{2}\beta^{-L-t}[1 - 2\beta^{-t-1}/(1 - \beta^{-t})]}$$

$$= \frac{-\frac{1}{2}[1 - 2\beta^{-1}/(\beta^{t} - 1)]}{\beta^{t-1} + \frac{1}{2}[1 - 2\beta^{-1}/(\beta^{t} - 1)]} \qquad (2.4.12)$$

$$> -\frac{1}{2}\beta^{1-t}[1 - 2\beta^{-1}/(\beta^{t} - 1)]$$

Although we have shown that (2.4.12) is a lower bound on $E_{x\oslash y}$ for all $\beta \geq 2$, $t \geq 1$; we have not found values of x and y for which (2.4.12) is precisely attained.

Consider case 2. From (2.4.6) and (2.4.11) it is evident that $E_{x\oslash y} > 0$. Moreover, since $\Delta r = \beta^{-L-t}$ and $\Delta q \geq \frac{1}{2}\beta^{-L-t}$ are identical to the expressions for Δr and Δp in case 2 of rounded multiplication (2.3.10), we obtain the same upper bound on $E_{x\oslash y}$ as that derived for case 2 of $E_{x\otimes y}$ (2.3.12).

Therefore, for rounded $x\oslash y$, we have shown (using cases 1 and 2) that

$$\frac{-\frac{1}{2}[1 - 2\beta^{-1}/(\beta^{t} - 1)]}{\beta^{t-1} + \frac{1}{2}[1 - 2\beta^{-1}/(\beta^{t} - 1)]} \leq E_{x\oslash y} \leq \frac{\frac{1}{2}}{\beta^{t-1} + \frac{1}{2}} \qquad (2.4.13)$$

for all $\beta \geq 2$, $t \geq 1$. We have not found values of x and y for which these bounds are precisely attained.

From (2.4.7) and (2.4.8), we see that the contents of the quotient register may assume values between β^{-2} and $1 - \beta^{-t}$. If, instead, the contents were allowed to take on values between β^{-1} and $\beta(1 - \beta^{-t})$, then pre-adjustment could be _eliminated_, and post-adjustment would require exactly 0 or 1 _right_-shifts. Alternatively, if it could be determined before quotient formation whether $x < y$ or $x \geq y$, then only $x \geq y$ need be right-shifted, the quotient register could be shortened to t digits, and no left-shifts would be required for post-adjustment.

If chopping is performed by rounding, then Δq must be compared with $\frac{1}{2}\beta^{-L-t}$. Probably the easiest way to do this is to extend the quotient register by an additional digit (in fact, a _bit_ is sufficient), and let the quotient formation process determine the $t+1$ st significant digit of the quotient.

The remarks comparing biased and unbiased rounding at the end of section 2.3 apply here equally well. We can distinguish between $\Delta q_1 = \Delta p_1$ and $\Delta q_2 = \Delta p_2$ (see 2.3.14) by comparing the remainder from the $t+1$ significant-digit quotient with zero. (No "sticky" bits are needed.)

As in section 2.3, all the error bounds derived for $x \oslash y$ indicate that base 2 would yield the smallest worst-case relative errors.

2.5 _Floating-Point Addition_

Consider $x \oplus y$ and $x \ominus y$, the fp sum and difference of the fp numbers x and y drawn from $\mathbb{F} = \mathbb{F}(\beta, t, M_1, M_2)$. Henceforth, we use the word _addition_ to denote either $\oplus$ or $\ominus$, and the word _sum_ to denote the result of either $\oplus$ or $\ominus$. Given that exponent overflow and underflow do not occur, we assume for any x and y in $\mathbb{F}$ that

$$x \oplus y = y \oplus x \qquad (2.5.0.1)$$

$$x \ominus y = x \oplus (-y) \tag{2.5.0.2}$$

$$x \ominus y = -(y \ominus x) \tag{2.5.0.3}$$

$$x \oplus 0 = x \tag{2.5.0.4}$$

$$y = -x \quad \Rightarrow \quad x \oplus y = 0 \tag{2.5.0.5}$$

Note that assumptions (2.5.0.1) - (2.5.0.5) are made, given that exponent overflow and underflow do not occur. Thus, for example, (2.5.0.3) need not hold for $y \ominus x$ = the most negative fp number in a 2's complement number system if negating the most negative fp number would result in exponent overflow. In a 2's complement number system where the most negative fp number is _defined_ to be the negative of the most positive fp number, exponent overflow cannot occur from negations, and (2.5.0.3) holds for all fp x and y such that $y \ominus x$ does not yield exponent overflow or underflow.

Also, note that assumptions (2.5.0.1) - (2.5.0.5) are made with respect to the _numbers_ that have fp representations, and do not necessarily apply to the fp _representations_ themselves. (Assumptions (2.5.0.3) and (2.5.0.4) do not always hold among the 1's complement representations of $x \ominus y$ and $y \ominus x$. For example, $+0 \oplus (-0) = -0$, and $x = y \Rightarrow [x \ominus y = -0$ & $-(y \ominus x) = +0]$.) These assumptions do hold for any reasonable implementation of β complement, $\beta - 1$ complement, or sign-magnitude arithmetic.

As in the case of fp multiplication and division considered in the previous sections, we shall be studying the maximum relative cropping errors in $\oplus$ and $\ominus$. For this purpose, assumptions (2.5.0.1) - (2.5.0.5) allow us, without loss of generality, to restrict our attention to x and y satisfying $x \geq y > 0$. Finally, since we shall be concerned only with relative errors, we assume without loss of generality, that the exponent of x (but not necessarily the exponent of y) is zero.

2.5.1 Chopped Addition: Scheme S_1

Suppose that $x \oplus y$ and $x \ominus y$ are formed as follows.

(1) Pre-Adjustment: y is right-shifted until its exponent equals the exponent of x, and then shifted y is chopped to $t+g$ digits. Let the resulting value be denoted y'.

(2) Addition: The sum ($x+y'$ or $x-y'$) is performed exactly in a $t+g$ digit accumulator with one overflow bit.

(3) Post-Adjustment: The sum ($x+y'$ or $x-y'$) is shifted left or right to place its leftmost nonzero digit in the most significant digit position in the accumulator. This shifted sum is then chopped to t digits.

In scheme S_1, $x \oplus y$ incurs two errors:

$\Delta y \equiv$ the error in representing y in $t+g$ digits, because of right-shifting y before the addition ($\Delta y \geq 0$);

$\Delta d \equiv$ the error in representing $x \oplus y$ in t digits, because the guard digits were discarded after the addition ($\Delta d \geq 0$).

It follows that $x \oplus y$ can be described by

$$\begin{aligned} x \oplus y &= x + (y - \Delta y) - \Delta d \\ &= (x + y) - (\Delta y + \Delta d) \end{aligned} \qquad (2.5.1.1)$$

and the relative error in $x \oplus y$ can be described by

$$E_{x \oplus y} \equiv \frac{x \oplus y - (x + y)}{(x + y)} = \frac{-(\Delta y + \Delta d)}{x \oplus y + (\Delta y + \Delta d)} \qquad (2.5.1.2)$$

Since $x \geq y > 0$, $\Delta y \geq 0$, and $\Delta d \geq 0$, we see that $E_{x \oplus y} \leq 0$. That 0 is a <u>least</u> upper bound on $E_{x \oplus y}$ is determined by observing $E_{x \oplus y} = 0$ when $x \oplus y = x + y$.

To obtain a lower bound on $E_{x \oplus y}$, recall that for $z > 0$ and $w \geq 0$, the function $f(w,z) \equiv w/(z + w)$ is maximized by minimizing z and maximizing w. Consequently, as in the case of chopped multiplication, a lower bound on $E_{x \oplus y}$ is obtained by substituting in (2.5.1.2) the smallest possible value for $x \oplus y$ and the largest possible value for $\Delta y + \Delta d$.

The carry-out of the leading digit in $x \oplus y$ can cause $x \oplus y$ to be shifted some number of positions, R, to the right during post-adjustment. Since during addition there can be at most one digit of carry-out from the most significant digit in the accumulator, it follows that for $x \geq y > 0$, $0 \leq R \leq 1$. For each such R, it is easy to see that the smallest possible value of $x \oplus y$ is β^{R-1}. Moreover, the absolute error in $x \oplus y$, namely $\Delta y + \Delta d$, will be as large as possible when $x \oplus y$ picks up the first t digits from an exact sum having $\beta - 1$ in positions $t + 1, \ldots, 2t$. Consequently, for each R, the largest possible value of $\Delta y + \Delta d$ is $\beta^R(\beta^{-t} - \beta^{-2t})$.

Summarizing, we have for $x \oplus y$,

$$\begin{aligned} E_{x \oplus y} &= \frac{-(\Delta y + \Delta d)}{x \oplus y + (\Delta y + \Delta d)} \geq \frac{-\beta^R(\beta^{-t} - \beta^{-2t})}{\beta^{R-1} + \beta^R(\beta^{-t} - \beta^{-2t})} \\ &= \frac{-(1 - \beta^{-t})}{\beta^{t-1} + (1 - \beta^{-t})} \qquad (2.5.1.3) \\ &> -(1 - \beta^{-t})\,\beta^{1-t} \end{aligned}$$

Bound (2.5.1.3) was derived independently by Thiran [63], who indicated that it is attained for all $\beta \geq 2$, $t \geq 1$, when

$$x = \beta^{-1} \quad \text{and} \quad y = (1 - \beta^{-t})\beta^{-t}$$

Consequently, (2.5.1.3) is a greatest lower bound on $E_{x \oplus y}$.

It is important to note that (2.5.1.3) is independent of g, the number of guard digits. Intuitively, any digits of y that are retained by the guard digit registers to participate in the fp sum of x and y are later discarded when the sum is chopped to t digits. Consequently, when x and y have the same sign, the chopped fp sum using 0 guard digits yields exactly the same result as if an infinite number of guard digits were available. In fact, the addition may be speeded up in this case by not allowing any guard digits to paricipate in the sum; hence, addition is not performed at all if the exponents of x and y differ by $\geq t$. This speedup is especially significant if addition is performed in an extended precision accumulator $(g \geq t)$.

Consider now $x \ominus y$. As in the case of $x \oplus y$, two errors are incurred. Borrowing the definitions of Δy and Δd from the discussion of $x \oplus y$, it follows that $x \ominus y$ can be described by

$$\begin{aligned} x \ominus y &= x - (y - \Delta y) - \Delta d \\ &= (x - y) + (\Delta y - \Delta d), \quad \Delta y \geq 0,\ \Delta d \geq 0 \qquad (2.5.1.4) \end{aligned}$$

and the relative error in $x \ominus y$ can be described by

$$E_{x \ominus y} \equiv \frac{x \ominus y - (x - y)}{(x - y)} = \frac{-(\Delta d - \Delta y)}{x \ominus y + (\Delta d - \Delta y)} \qquad (2.5.1.5)$$

Assumptions (2.5.0.2) and (2.5.0.5) allow us to consider only $x \neq y$ in $x \ominus y$. Thus, for $x > y > 0$, any lower bound on $E_{x \ominus y}$ must occur when $\Delta d \geq \Delta y$, and any upper bound on $E_{x \ominus y}$ must occur when $\Delta y \geq \Delta d$. If $\Delta y = \Delta d$, then $x \ominus y$ is exact, so we may neglect this possibility without loss of generality.

Suppose $\Delta d > \Delta y \geq 0$. Recall that, for $z > 0$ and $w \geq 0$, the function $f(w,z) \equiv w/(z+w)$ is maximized by minimizing z and maximizing w. Consequently, a lower bound on $E_{x\ominus y}$ is obtained by substituting in (2.5.1.5) the smallest possible value for $x\ominus y$ and the largest possible value for $\Delta d - \Delta y$.

The minimum value of $x\ominus y$ is obtained by subtracting from the smallest possible x the largest y satisfying $x > y > 0$. Thus,

$$x \ominus y \geq \beta^{-1} - (\beta^{-1} - \beta^{-t-1}) = \beta^{-t-1} \qquad (2.5.1.6)$$

Let L be the number of leading zeros in $x - y'$ prior to post-adjustment. From (2.5.1.6) we see that $0 \leq L \leq t$, for $x > y > 0$. For each such L, the smallest possible value of $x\ominus y$ is β^{-L-1}. Moreover, the absolute error in $x\ominus y$, namely $\Delta d - \Delta y$, will be as large as possible when $\Delta y = 0$ and each of the guard digits equals $\beta - 1$. Consequently, for each L, $\Delta d - \Delta y$ can be no larger than $\beta^{-L}(\beta^{-t} - \beta^{-t-g})$.

Summarizing, we have

$$E_{x\ominus y} = \frac{-(\Delta d - \Delta y)}{x \ominus y + (\Delta d - \Delta y)} \geq \frac{-\beta^{-L}(\beta^{-t} - \beta^{-t-g})}{\beta^{-L-1} + \beta^{-L}(\beta^{-t} - \beta^{-t-g})}$$

$$= \frac{-(1 - \beta^{-g})}{\beta^{t-1} + (1 - \beta^{-g})} \qquad (2.5.1.7)$$

$$> -(1 - \beta^{-g})\,\beta^{1-t} \qquad (2.5.1.8)$$

One can verify that bound (2.5.1.7) is indeed attained when

$$x = \beta^{-1} + \beta^{-t} \quad \text{and} \quad y = \beta^{-t-g}$$

Consequently, (2.5.1.7) is a <u>greatest</u> lower bound on $E_{x\ominus y}$.

Note that if $g = 0$ then $\Delta d = 0$. This case is not handled here, because we are considering the case where $\Delta d > \Delta y \geq 0$. Bound (2.5.1.7) does not appear in Thiran's paper [63] since he considered the error in $x \oplus y$ together with the error in $x \ominus y$, and for $g < t$, (2.5.1.3) is a <u>smaller</u> lower bound than (2.5.1.7). However, it is important to consider (2.5.1.7) because it shows a dependence upon g, whereas (2.5.1.3) does not. In fact, (2.5.1.7) tells us to <u>minimize</u> the number of guard digits in order to minimize the negative relative error in $x \ominus y$, and (2.5.1.3) tells us that by choosing the minimum number of guard digits, we will <u>neither increase nor decrease</u> the worst-case relative error in $x \oplus y$! (Recall, $E_{x \oplus y} \le 0$, independent of g.) But, before we can settle upon a particular value of g, we must consider also an upper bound on the (positive) relative error in $E_{x \ominus y}$.

Suppose that $\Delta y > \Delta d \ge 0$. Let q be the number of digit positions y is shifted to the right before forming $x \ominus y$. Since the exponent of $x = 0$ and the minimum exponent for $y = -M_1$, it follows that $0 \le q \le M_1$. Note that if $0 \le q \le g$, then $\Delta y = 0$. This case is not handled here, because we are considering the case where $\Delta y > \Delta d \ge 0$. For each q such that $0 \le g < q \le M_1$, the smallest possible value of $x - y$ is obtained by subtracting the largest fp number with exponent $= -q$ from the smallest possible value of x. Thus, $\min(x - y) = \beta^{-1} - \beta^{-q}(1 - \beta^{-t})$.

The largest possible value of $\Delta y - \Delta d$ is obtained when $\Delta d = 0$, and $q - g$ digits of y with size $\beta - 1$ have been right-shifted beyond the rightmost guard digit register. That is, for each q, $\Delta y - \Delta d$ can be no larger than $\beta^{-t-g} - \beta^{-t-q}$. Finally, since y can contain no more than t nonzero digits, we need consider only $0 \le q - g \le t$; i.e. it is sufficient to consider $0 \le g < q \le \min(g + t, M_1)$.

Summarizing, we have

$$E_{x\ominus y} = \frac{(\Delta y - \Delta d)}{x - y} \le \frac{\beta^{-t-g} - \beta^{-t-q}}{\beta^{-1} - \beta^{-q}(1 - \beta^{-t})}$$

$$= \frac{\beta^{-t}(\beta^{q-g} - 1)}{(\beta^{q-1} - 1) + \beta^{-t}} \qquad (2.5.1.9)$$

$$= \frac{1}{\beta^{t+g-1} + [\beta^{t}(\beta^{g-1} - 1) + 1]/(\beta^{q-g} - 1)} \qquad (2.5.1.10)$$

For $g = 0$ and $q = 1$, (2.5.1.9) leads to $E_{x\ominus y} \le \beta - 1$. This bound is, in fact attained when

$$x = \beta^{-1} \quad \text{and} \quad y = \beta^{-1}(1 - \beta^{-t})$$

Thus, if no guard digits are used to form $x\ominus y$, then the relative error may be as large as $(\beta - 1) \times 100$ percent! Note well that this worst case error will not at all be affected by changing t, the number of digits in the fp operands. According to the characterization of the fp arithmetic given in [26,41], an error of precisely this magnitude is possible on the Univac 1100 series computers with single-precision numbers and on the CDC 6000/Cyber-70 series computers with double precision numbers.

We shall take the point of view that forming $x\ominus y$ requires at least one guard digit. For $g \ge 1$, (2.5.1.10) is maximized when q is a maximum. Since we are assuming $q \le \min(g + t, M_1)$, we make the additional reasonable assumption that $g + t \le M_1$, and substitute $g + t$ for q in (2.5.1.9). This yields

$$E_{x\ominus y} \le \beta^{-t} \frac{(\beta^{t} - 1)}{(\beta^{t+g-1} - 1) + \beta^{-t}} \qquad (2.5.1.11)$$

$$= \frac{(1 - \beta^{-t})}{\beta^{t+g-1} - (1 - \beta^{-t})} \qquad (2.5.1.12)$$

Bound (2.5.1.12), except for a missing "-" in β^{-t} in the denominator of (2.5.1.12) (probably the result of a typing error), was derived independently by Thiran [63]. He indicated that this bound is attained when

$$x = \beta^{-1} \quad \text{and} \quad y = \beta^{-t-g}(1 - \beta^{-t})$$

Consequently, (2.5.1.12) is a <u>least</u> upper bound on $E_{x \ominus y}$.

For $x \ominus y$, observe that the maximum negative relative error (2.5.1.7) is minimized by <u>minimizing</u> the number of guard digits, g, and that the maximum positive relative error (2.5.1.12) is minimized by <u>maximizing</u> the number of guard digits. Also, note that the magnitude of (2.5.1.7) is strictly <u>greater</u> than the magnitude of (2.5.1.12) for every choice of $\beta \geq 2$, $t \geq 1$, and $g \geq 1$, <u>except</u> when $\beta = 2$ and $g = 1$. Thus, except for the latter, exceptional case, it follows that the optimal value of $g \geq 1$ for minimizing the maximum relative error in $x \ominus y$ is $g = 1$.

For $x \ominus y$ where $\beta = 2$, (2.5.1.7) and (2.5.1.12) tell us that the magnitude of the maximum positive relative error with $g = 1$ is $\geq$ the magnitude of the maximum negative relative error with $g = 1$, but $<$ the magnitude of the maximum negative relative error with $g \geq 2$. That is,

$$\frac{(1 - 2^{-1})}{2^{t-1} + (1 - 2^{-1})} \leq \frac{(1 - 2^{-t})}{2^{t} - (1 - 2^{-t})} < \frac{(1 - 2^{-g})}{2^{t-1} + (1 - 2^{-g})}$$

for $g \geq 2$ and $t \geq 1$. Consequently, the optimal value of $g \geq 1$ for minimizing the maximum relative error in $x \ominus y$ when $\beta = 2$ is also $g = 1$, even though in this case, the maximum relative error is given by (2.5.1.12) instead of (2.5.1.7).

The choice of one guard digit is also beneficial for $x \oplus y$, since it makes the carry-propagate delay small, it allows the same hardware to be used for $x \oplus y$ as $x \ominus y$, and, as we saw in the previous section, it has no effect upon the maximum relative error in x⊕y.

The experiments of Kuki and Cody [44] show that the _average number of significant digits_ in the sum of mixed-sign summands is largest for chopped arithmetic when the number of guard digits is smallest. They observe that in the case of mixed-sign summands, "the absence of guard digits...actually helps to neutralize round-off errors....If guard digits are used...the sum will statistically tend toward zero [have a large negative error] regardless of the sign combination....This tendency becomes more pronounced as more guard digits are used." Kuck, et. al. [43] show analytically that the _average bias_ in representing real numbers as fp numbers is minimized by minimizing the number of guard digits. They observe that the error in fp addition can be considered as just the error in representing the exact sum of two fp numbers. Hence, they argue that the average bias in the fp sum is also minimized by minimizing the number of guard digits, consistent with the statistical experiments of Kuki and Cody. Thus, from the standpoint of accuracy, efficiency, and hardware cost, the choice of one guard digit for performing chopped addition is best.

An important consequence of the preceding paragraph is that, for chopped arithmetic, the fp sum has a greater worst-case error, a greater average error (measured by the number of significant digits), and a greater bias when an extended precision accumulator ($g \geq t$) is used than when it is not. Although this fact follows directly from the work by Kuki and Cody [44] and the work by Kuck et. al. [43], it appears that this fact is not generally known.

Thiran [63] concludes that, "a single accumulator with one guard digit and a double length accumulator give rise to exactly the same maximum error", because he uses the same maximum error bound to bound the error when the summands have like signs as when the signs differ. Kaneko and Liu [40] conclude that, "even with one guard digit, a single-precision accumulator can perform addition almost as accurately as a double-precision accumulator if the radix is moderately large". Not only have we shown that one guard digit yields <u>more accurate</u> results than a double-precision accumulator, but since the worst-case error in $\oplus$ and $\ominus$ is given by (2.5.1.3), we see that the maximum relative error is smallest when the radix is as <u>small as possible</u>, namely $\beta = 2$. In fact, apparently, not even Kuck et. al. [43], fully appreciate the superior performance of using just one guard digit for chopped addition, since their preferred scheme employs a special "sticky" bit <u>with</u> one guard digit to produce the same results as if an infinite precision accumulator was used.

We showed in the previous section that if the exponents of x and y differ by t or more, then $x \oplus y = x$, and thus $x \oplus y$ need not be explicitly formed by the hardware. We now specify conditions under which <u>less accurate</u> results would occur if $x \ominus y$ were actually formed than if it were not. Suppose that the exponents of x and y differ by $q \geq 0$. (We have been assuming without loss of generality that the exponent of x equals 0, and the exponent of y is strictly smaller than the exponent of x. Hence, we now let the exponent of y be $-q$.) During pre-adjustment, the leading digit of y is shifted q places to the right, leaving q leading zeros in the fractional part of y. If $t+1 \leq q \leq t+g-1$, then during subtraction a borrow is obtained from the low-order nonzero digit in x, which will leave digits of size $\beta - 1$ in positions $t+1, \ldots, q$ to the right of the radix in the fp sum. If $x \neq \beta^{-1}$, then the leading digit of the sum will be nonzero and no post-adjustment shift is necessary. We have noted that digit position $t+1$ contains $\beta - 1$; consequently, the relative error incurred by chopping after the t-th digit of the sum is at least

$$\beta^{-t-1}(\beta - 1)/(x - y) \qquad (2.5.1.13)$$

The error would have been only

$$y/(x - y) \le \beta^{-t-1}(1 - \beta^{-t})/(x - y)$$

if subtraction had not been performed. If $x = \beta^{-1}$, then there will be exactly one leading zero digit in the fp sum, necessitating exactly one shift during post-adjustment. (Digit positions $2, \ldots, q$ will be of size $\beta - 1$.) In this case, the digit that was in position $t+2$ of the sum prior to post-adjustment is found in position $t+1$ after post-adjustment. Thus, if $t+2 \le q \le t+g-1$, it follows from our previous remarks that position $t+1$ of the post-adjusted sum will contain a digit of size $\beta - 1$. The relative error incurred by chopping after the t-th digit of the sum is once again given by (2.5.1.13), whereas the (smaller) error that would be incurred if subtraction had not been performed is

$$y/(x - y) \le \beta^{-t-2}(1 - \beta^{-t})/(x - y)$$

For completeness, it should be noted that $t+1 \le q \le t+g-1$ is not vacuous when $g \ge 2$, and $t+2 \le q \le t+g-1$ is not vacuous when $g \ge 3$. Thus, these remarks apply to most extended precision accumulators ($g \ge t$) as a special case.

Finally, we note that Oliver [55] also published bounds on the maximum relative error in scheme S_1 at about the same time as Thiran. However, his bounds are not as sharp as the bounds derived here or given by Thiran.

2.5.2 Rounded Addition: Scheme S_2

Suppose that $x \oplus y$ and $x \ominus y$ are formed as follows.

(1) Pre-Adjustment: y is right-shifted until its exponent equals the exponent of x, and then the shifted y is rounded to $t+g$ digits. Denote the resulting value by y'.

(2) Addition: The sum ($x+y'$ or $x-y'$) is performed exactly in a $t+g$ digit accumulator with one overflow bit.

(3) Post-Adjustment: The sum ($x+y'$ or $x-y'$) is shifted left or right to place its leftmost nonzero digit in the most significant digit position in the accumulator. This shifted sum is then rounded to t digits.

Formally,

$$\begin{aligned} x \oplus y &= x + (y - \Delta y + \Delta r_1) - \Delta d + \Delta r_2 \\ &= (x + y) + (\Delta r_1 - \Delta y) + (\Delta r_2 - \Delta d) \qquad (2.5.2.1) \end{aligned}$$

$$\begin{aligned} x \ominus y &= x - (y - \Delta y + \Delta r_1) - \Delta d + \Delta r_2 \\ &= (x - y) - (\Delta r_1 - \Delta y) + (\Delta r_2 - \Delta d) \qquad (2.5.2.2) \end{aligned}$$

where,

$$\begin{aligned} \Delta r_1 &= 0 && \text{if } \Delta y < \tfrac{1}{2}\beta^{-t-g} \\ \Delta r_1 &= \beta^{-t-g} && \text{if } \Delta y \ge \tfrac{1}{2}\beta^{-t-g} \end{aligned}$$

$$\begin{aligned} \Delta r_2 &= 0 && \text{if } \Delta d < \tfrac{1}{2}\beta^{-L-t} \\ \Delta r_2 &= \beta^{-L-t} && \text{if } \Delta d \ge \tfrac{1}{2}\beta^{-L-t} \end{aligned}$$

L is the number of digit positions $x+y'$ or $x-y'$ is left-shifted during post-adjustment ($-1 \le L \le t$), and the definitions of Δy and Δd are the same as for chopped addition ($\Delta y \ge 0$, $\Delta d \ge 0$).

Scheme S_2 was used by Wilkinson [70,71] for his a priori error analyses, with $g = 0$ (single-precision accumulator) and $g = t$ (double-precision accumulator). He did not give a posteriori bounds on the relative error in $\oplus$ or $\ominus$ for $g = 0$, but the fp difference between the following fp numbers shows that a relative error of <u>100 percent</u> is attainable, with any $\beta \geq 2$.

$$x = \beta^{-1} \quad \text{and} \quad y = \beta^{-1} - \frac{1}{2}\beta^{-t} \tag{2.5.2.3}$$

(Although the rounding schemes used on the CDC 3000 and CDC 6000/Cyber-70 series computers [26,41] are somewhat different than scheme S_2, it is instructive to note that these popular computers produce identically the same results and errors in forming the rounded fp difference indicated by (2.5.2.3) as scheme S_2 with $g = 0$!)

Wilkinson claimed that with $g = t$ the relative error in $\oplus$ and $\ominus$ could be bounded as follows.

$$|E_{x\oplus y}|,\ |E_{x\ominus y}| \leq \frac{1}{2}\beta^{1-t} \quad \text{if} \quad g = t \tag{2.5.2.4}$$

Later, Kaneko and Liu [40] showed that if $1 \leq g \leq t$, then Wilkinson's bound (2.5.2.4) for scheme S_2 should be increased by a factor of $1/(1 - \beta^{-g})$. Their bounds follow.

$$|E_{x\oplus y}|,\ |E_{x\ominus y}| \leq \frac{1}{2}\beta^{1-t}/(1 - \beta^{-g}) \quad \text{if} \quad 1 \leq g \leq t \tag{2.5.2.5}$$

Apparently, Thiran [63] decided that (2.5.2.5) was an unsatisfactorily "loose" bound, since he subsequently derived the following precise bounds for scheme S_2.

Let

$$B_1 \equiv \frac{\frac{1}{2}(1 + \beta^{-g})}{\beta^{t-1} + \frac{1}{2}(1 - \beta^{-g})} \tag{2.5.2.6}$$

$$B_2 = \frac{\frac{1}{2}}{\beta^{t-1} + \frac{1}{2}} \qquad (2.5.2.7)$$

$$B_3 = \frac{\frac{1}{2}(1 - \beta^{-g-1} - 2\beta^{-t})}{\beta^{t-1} + \frac{1}{2}(1 - \beta^{-g-1} - 2\beta^{-t})} \qquad (2.5.2.8)$$

$$B_4 = \frac{\frac{1}{2}(1 - 2\beta^{-t})}{\beta^{t-1} + \frac{1}{2}(1 - 2\beta^{-t})} \qquad (2.5.2.9)$$

Now, for $\beta > 2$,

$$-B_3 \le E_{x\oplus y}, E_{x\ominus y} \le B_1 \quad \text{if} \quad 1 \le g \le t-2 \qquad (2.5.2.10)$$

$$-B_4 \le E_{x\oplus y}, E_{x\ominus y} \le B_1 \quad \text{if} \quad g = t-1 \qquad (2.5.2.11)$$

$$-B_4 \le E_{x\oplus y}, E_{x\ominus y} \le B_2 \quad \text{if} \quad g = t \qquad (2.5.2.12)$$

and for $\beta = 2$,

$$-B_2 \le E_{x\oplus y}, E_{x\ominus y} \le B_1 \quad \text{if} \quad 1 = g \qquad (2.5.2.13)$$

$$-B_3 \le E_{x\oplus y}, E_{x\ominus y} \le B_1 \quad \text{if} \quad 2 \le g \le t-2 \qquad (2.5.2.14)$$

$$-B_4 \le E_{x\oplus y}, E_{x\ominus y} \le B_1 \quad \text{if} \quad t-1 \le g \le t \qquad (2.5.2.15)$$

(Oliver [55] also published bounds for scheme S_2 at about the same time as Thiran. However, Oliver's bounds are not as sharp as possible.)

Thiran pointed out that Kaneko and Liu's bound (2.5.2.5) is looser than necessary for all $\beta \ge 2$ and $1 \le g \le t$. He also mentioned that Wilkinson's bound (2.5.2.4) only bounds the error for $g = t$ when $\beta > 2$. If $\beta = 2$, the greater upper bound, B_1, is attained by $x\oplus y$, where

$$x = 2^{-1} \quad \text{and} \quad y = (1 - 2^{-t})\,2^{-t-1} \qquad (2.5.2.16)$$

In fact, by closely examining (2.5.2.10) - (2.5.2.15), one sees that the bounds for $\beta > 2$ and $\beta = 2$ differ only when $g = 1$ or $g = t$. When $g = 1$, the fp difference between the numbers

$$x = \beta^{-1} \quad \text{and} \quad y = (1 - \tfrac{1}{2}\beta^{1-t})\,\beta^{-2}$$

yields a negative relative error of size

$$E_{x\ominus y} = \frac{-\frac{1}{2}}{(\beta - 1)\,\beta^{t-1} + \frac{1}{2}} \qquad (2.5.2.17)$$

which has a larger magnitude than B_3 only if $\beta = 2$. That is, only when $\beta = 2$ do we have $\beta - 1 = 1$ in (2.5.2.17). (Note that $t \geq 3$ is required for bound B_3 to be applicable.) When $g = t$, the positive relative error in $x \ominus y$ requires special treatment for the case of $\beta = 2$, because only when $\beta = 2$ is it possible to find an fp number y such that rounding the 2t-th digit of y during pre-adjustment will cause a carry to propagate into the $(t+1)$ st digit of y, thereby making y a candidate for an additional round during post-adjustment. (The y given in (2.5.2.16) is one fp number that enjoys this property.) Thus, only when $\beta = 2$ do we have $1\,(\text{carry}) = \frac{1}{2}\beta\,(\text{criterion for rounding})$.

Wilkinson's oversight in regard to (2.5.2.4) with $\beta = 2$ can probably be traced to his (erroneous) statement [70, p. 8], "If $b_1 - b_2 > t$ then x_2 is too small to have any effect as far as the first t significant digits of the sum are concerned, and we have

$$fl(x_1 + x_2) \equiv x_1 \text{ "}$$

(In his notation, b_1 is the exponent of x_1, b_2 is the exponent of x_2, and $fl(x_1 + x_2)$ is the fp sum of the signed fp numbers, x_1, and x_2.) Wilkinson's statement is false not only for $\beta = 2$ with the fp sum of

(2.5.2.16), but, without making reference to Wilkinson, Knuth [42, section 4.2.1, exercise 5] has shown that it is false for *any* $\beta \geq 2$ with $x \ominus y$, where

$$x = \beta^{-1} \quad \text{and} \quad y = (\tfrac{1}{2} + \beta^{-t})\,\beta^{-t-1}$$

2.5.3 *Rounded Addition*: Scheme S_3

Suppose that $x \oplus y$ and $x \ominus y$ are formed as follows.

(1) *Pre-Adjustment*: y is right-shifted until its exponent equals the exponent of x, and then the shifted y is *chopped* to $t+g$ digits. Denote the resulting value by y'.

(2) *Addition*: (Same as scheme S_2)

(3) *Post-Adjustment*: (Same as scheme S_2)

Formally,

$$\begin{aligned} x \oplus y &= x + (y - \Delta y) - \Delta d + \Delta r \\ &= (x + y) - (\Delta d + \Delta y) + \Delta r \qquad (2.5.3.1) \\ x \ominus y &= x - (y - \Delta y) - \Delta d + \Delta r \\ &= (x - y) - (\Delta d - \Delta y) + \Delta r \qquad (2.5.3.2) \end{aligned}$$

where,

$$\begin{aligned} \Delta r &= 0 & \text{if} \quad \Delta d &< \tfrac{1}{2}\beta^{-L-t} \\ \Delta r &= \beta^{-L-t} & \text{if} \quad \Delta d &\geq \tfrac{1}{2}\beta^{-L-t} \end{aligned}$$

L is the number of digit positions $x + y'$ or $x - y'$ is left-shifted during post-adjustment ($-1 \leq L \leq t$), and the definitions of Δy and Δd are the same as for chopped addition ($\Delta y \geq 0$, $\Delta d \geq 0$).

The relative error in $x \oplus y$ and $x \ominus y$ is given by

$$E_{x\oplus y} = \frac{x \oplus y - (x + y)}{(x + y)} = \frac{\Delta r - (\Delta d + \Delta y)}{(x \oplus y - \Delta r) + (\Delta d + \Delta y)} \quad (2.5.3.3)$$

$$F_{x\ominus y} = \frac{x \ominus y - (x - y)}{(x - y)} = \frac{\Delta r - (\Delta d - \Delta y)}{(x \ominus y - \Delta r) + (\Delta d - \Delta y)} \quad (2.5.3.4)$$

If $g = 0$, then it is clear that rounding in scheme S_3 can only take place when $\oplus$ yields a carry into the overflow bit. (Post-adjustment rounding has no effect for $\ominus$, since with $g = 0$ the number of nonzero digits in $x\ominus y$ will always be $\leq t$.) Moreover, the maximum relative error will be attained in the same case that has given rise to the maximum relative error for chopped addition with $g = 0$ (2.5.1.9). Thus, if no guard digits are used for scheme S_3, then the maximum relative error will be $(\beta - 1) \times 100$ percent, as we saw for chopped addition. Henceforth, we shall assume that $g \geq 1$.

Consider $x\oplus y$. We investigate the nonzero error in $x\oplus y$ with the following separate, but exhaustive cases. (Refer to 2.5.3.1.)

Case 1. $\Delta d < \frac{1}{2}\beta^{-L-t}$ and $\Delta d + \Delta y > \Delta r = 0$ (2.5.3.5)

Case 2. $\Delta d \geq \frac{1}{2}\beta^{-L-t}$ and $\Delta d + \Delta y < \Delta r = \beta^{-L-t}$ (2.5.3.6)

Case 3. $\Delta d \geq \frac{1}{2}\beta^{-L-t}$ and $\Delta d + \Delta y > \Delta r = \beta^{-L-t}$ (2.5.3.7)

Consider case 1. ($\Delta r = 0$) Using reasoning analogous to that employed in the previous sections, it is evident that $E_{x\oplus y}$ may be bounded below (for case 1) by substituting in (2.5.3.3) the smallest possible value for $x\oplus y$ and the largest possible value for $\Delta d + \Delta y$.

Since we have defined L to be the number of positions $x + y'$ is shifted to the left during post-adjustment, and since a carry-out from the most significant digit in the accumulator requires a right-shift during post-adjustment, it follows that $-1 \leq L \leq 0$.

From (2.5.3.5) we see $\Delta d < \frac{1}{2}\beta^{-L-t}$. But, since Δd only accounts for error to the left of the last guard digit, and $g \geq 1$, we have

$$\Delta d \leq \frac{1}{2}\beta^{-L-t} - \beta^{-t-g} \qquad (2.5.3.8)$$

Likewise, Δy only accounts for error to the right of the last guard digit. Hence,

$$\Delta y < \beta^{-t-g} \qquad (2.5.3.9)$$

Combining (2.5.3.8) and (2.5.3.9), we obtain the following bound on the total absolute error in $x \oplus y$.

$$\Delta d + \Delta y < \frac{1}{2}\beta^{-L-t}$$

But, since y can have no more than t nonzero digits, it follows that the absolute error in $x \oplus y$ can contain no more than t nonzero digits. Consequently,

$$\Delta d + \Delta y \leq \frac{1}{2}\beta^{-L-t} - \beta^{-L-2t} \qquad \text{if } \beta > 2$$

$$\Delta d + \Delta y \leq 2^{-L-t-1} - 2^{-L-2t-1} \qquad \text{if } \beta = 2$$

For each L, $-1 \leq L \leq 0$, the smallest possible value of $x \oplus y$ is β^{-L-1}.

Summarizing for case 1 with $\beta > 2$,

$$E_{x\oplus y} = \frac{-(\Delta d + \Delta y)}{x \oplus y + (\Delta d + \Delta y)} \geq \frac{-\beta^{-L}(\frac{1}{2}\beta^{-t} - \beta^{-2t})}{\beta^{-L-1} + \beta^{-L}(\frac{1}{2}\beta^{-t} - \beta^{-2t})}$$

$$= \frac{-(\frac{1}{2} - \beta^{-t})}{\beta^{t-1} + (\frac{1}{2} - \beta^{-t})} \qquad (2.5.3.10)$$

$$> -(\tfrac{1}{2} - \beta^{-t})\,\beta^{1-t}$$

And, for case 1 with $\beta = 2$,

$$E_{x\oplus y} = \frac{-(\Delta d + \Delta y)}{x \oplus y + (\Delta d + \Delta y)} \geq \frac{-2^{-L-1}(2^{-t} - 2^{-2t})}{2^{-L-1} + 2^{-L-1}(2^{-t} - 2^{-2t})}$$

$$= \frac{-(\tfrac{1}{2} - 2^{-t-1})}{2^{t-1} + (\tfrac{1}{2} - 2^{-t-1})} \qquad (2.5.3.11)$$

$$> -(\tfrac{1}{2} - 2^{-t-1})\,2^{1-t}$$

Note that (2.5.3.10) is attained by $x \oplus y$ when

$$x = \beta^{-1} \quad \text{and} \quad y = (\tfrac{1}{2} - \beta^{-t})\,\beta^{-t}$$

and (2.5.3.11) is attained by $x \oplus y$ when

$$x = \tfrac{1}{2} \quad \text{and} \quad y = (\tfrac{1}{2} - 2^{-t})\,2^{-t-1}$$

Note also that bounds (2.5.3.10) and (2.5.3.11) are independent of g, the number of guard digits.

Consider case 2. ($\Delta r = \beta^{-L-t}$) Recall that L is the number of digit positions that $x + y'$ can be shifted to the left during post-adjustment, but prior to rounding. (Equivalently, L is the number of leading zeros in $x + y'$.) Thus, prior to post-adjustment rounding, for each L, $-1 \leq L \leq 0$, the fp sum must be at least β^{-L-1}; i.e., $x \oplus y - \Delta r \geq \beta^{-L-1}$. Since $\Delta d \geq \frac{1}{2}\beta^{-L-t}$, from (2.5.3.1) we obtain

$$x + y = (x \oplus y - \Delta r) + (\Delta d + \Delta y) \geq \beta^{-L-1} + \tfrac{1}{2}\beta^{-L-t}$$

By (2.5.3.6) we know that $\Delta d + \Delta y < \Delta r$. Thus for each L, the largest possible value of $\Delta r - (\Delta d + \Delta y)$ occurs when $\Delta y = 0$ and $\Delta d = \frac{1}{2}\beta^{-L-t}$.

Summarizing for case 2 with $\beta \geq 2$,

$$E_{x\oplus y} = \frac{\Delta r - (\Delta d + \Delta y)}{(x + y)} \leq \frac{\frac{1}{2}\beta^{-L-t}}{\beta^{-L-1} + \frac{1}{2}\beta^{-L-t}}$$

$$= \frac{\frac{1}{2}}{\beta^{t-1} + \frac{1}{2}} \qquad (2.5.3.12)$$

$$< \frac{1}{2}\beta^{1-t}$$

One can verify that (2.5.3.12) is indeed attained by $x\oplus y$ when

$$x = 1 - \beta^{-t} \quad \text{and} \quad y = (1 + \tfrac{1}{2}\beta)\,\beta^{-t}$$

Once again, note that (2.5.3.12) is independent of g, the number of guard digits.

Consider case 3. From the discussion preceding (2.5.1.3), we know that the largest possible value of $\Delta d + \Delta y$ is

$$\max(\Delta d + \Delta y) = \beta^{-L}(\beta^{-t} - \beta^{-2t}) < \beta^{-L-t}$$

Consequently, case 3 cannot arise.

Consider now $x\ominus y$. Assumptions (2.5.0.1) - (2.5.0.5) allow us, without loss of generality, to restrict our attention to x and y satisfying $x > y > 0$. We investigate the nonzero error in $x\ominus y$ with the following separate, but exhaustive cases. (Refer to 2.5.3.2.)

Case 1. $\Delta d < \frac{1}{2}\beta^{-L-t}$ and $\Delta y + 0 < \Delta d$ (2.5.3.13)

Case 2. $\Delta d < \frac{1}{2}\beta^{-L-t}$ and $\Delta y + 0 > \Delta d$ (2.5.3.14)

Case 3. $\quad \Delta d \;\geq\; \frac{1}{2}\beta^{-L-t} \quad$ and $\quad \Delta y + \beta^{-L-t} \;<\; \Delta d$ (2.5.3.15)

Case 4. $\quad \Delta d \;\geq\; \frac{1}{2}\beta^{-L-t} \quad$ and $\quad \Delta y + \beta^{-L-t} \;>\; \Delta d$ (2.5.3.16)

Consider case 1. ($\Delta r = 0$) Using reasoning analogous to that employed in the previous sections, we see that $E_{x\ominus y}$ may be bounded below by substituting in (2.5.3.4) the smallest possible value for $x\ominus y$ and the largest possible value for $\Delta d - \Delta y$.

From (2.5.1.16) it is evident that $0 \leq L \leq t$. For each such L, the smallest possible value of $x\ominus y$ is β^{-L-1}. From (2.5.3.13) we have $\Delta d < \frac{1}{2}\beta^{-L-t}$. But, since Δd only accounts for error to the left of the last guard digit, and $g \geq 1$, it follows that $\Delta d \leq \beta^{-L}(\frac{1}{2}\beta^{-t} - \beta^{-t-g})$. Moreover, $\Delta d - \Delta y$ is maximized when $\Delta y = 0$.

Summarizing for case 1 with $\beta \geq 2$,

$$E_{x\ominus y} = \frac{-(\Delta d - \Delta y)}{x \ominus y + (\Delta d - \Delta y)} \geq \frac{-\beta^{-L}(\frac{1}{2}\beta^{-t} - \beta^{-t-g})}{\beta^{-L-1} + \beta^{-L}(\frac{1}{2}\beta^{-t} - \beta^{-t-g})}$$

$$= \frac{-(\frac{1}{2} - \beta^{-g})}{\beta^{t-1} + (\frac{1}{2} - \beta^{-g})} \qquad (2.5.3.17)$$

$$> -(\tfrac{1}{2} - \beta^{-g})\,\beta^{1-t}$$

One can verify that (2.5.3.17) is indeed attained when

$$x = \beta^{-1} + \beta^{-t} \quad \text{and} \quad y = (\tfrac{1}{2} + \beta^{-g})\,\beta^{-t}$$

By examining (2.5.3.17), we see that the negative relative error applicable to case 1 is minimized by <u>minimizing</u> g, the number of guard digits.

Consider case 2. From (2.5.3.14), note that $\Delta r = 0$ and $\Delta y > \Delta d \geq 0$.

Thus, the maximum relative error applicable to case 2 is identical to that attributed to chopped addition when $\Delta y > \Delta d \geq 0$. (Refer to 2.5.1.9 - 2.5.1.12.)

Consider case 3. From the discussion preceding (2.5.1.7), we see that the largest possible value of Δd is

$$\max \Delta d = \beta^{-L}(\beta^{-t} - \beta^{-t-g}) < \beta^{-L-t}$$

Consequently, case 3 cannot arise.

Consider case 4. ($\Delta r = \beta^{-L-t}$) Let q be the number of digit positions y is shifted to the right during pre-adjustment. From the discussion preceding (2.5.1.9), we see that $0 \leq q \leq M_1$.

Suppose $q \leq g$. Note that this implies $\Delta y = 0$. Prior to post-adjustment rounding, for each L, $0 \leq L \leq t$, the fp difference must be at least β^{-L-1}. I.e., $x \ominus y - \Delta r \geq \beta^{-L-1}$. From (2.5.3.16) we have $\Delta d \geq \frac{1}{2}\beta^{-L-t}$. Thus, for each L,

$$\Delta r - \Delta d \leq \beta^{-L-t} - \frac{1}{2}\beta^{-L-t} = \frac{1}{2}\beta^{-L-t}$$

Summarizing for case 4 with $q \leq g$ and $\beta \geq 2$,

$$E_{x\ominus y} = \frac{(\Delta r - \Delta d)}{(x \ominus y - \Delta r) + \Delta d} \leq \frac{\frac{1}{2}\beta^{-L-t}}{\beta^{-L-1} + \frac{1}{2}\beta^{-L-t}}$$

$$= \frac{\frac{1}{2}}{\beta^{t-1} + \frac{1}{2}} \qquad (2.5.3.18)$$

(Note that (2.5.3.18) is independent of g.) One can verify that (2.5.3.18) is indeed attained when

$$x = \beta^{-1} + \beta^{-t} \quad \text{and} \quad y = \frac{1}{2}\beta^{-t}$$

Suppose now $q > g$. If $L = 0$, then prior to post-adjustment rounding, the fp difference must be at least β^{-1}. I.e., $x \ominus y - \Delta r \geq \beta^{-1}$. If $L \neq 0$ and $g = 1$, then during post-adjustment the contents of the single guard digit is shifted left. Consequently, $\Delta d = 0$ and no rounding is performed; hence, case 4 does not apply.

Maintaining the assumption that $q > g$, let us now consider $g \geq 2$. Since $q \geq 3$, then prior to post-adjustment rounding, the fp difference must be at least $\beta^{-1} - \beta^{-3}$. I.e., $x \ominus y - \Delta r \geq \beta^{-1} - \beta^{-3}$. Note that this implies $L \leq 1$.

For $q > g$, the largest possible value of Δy is obtained when $q - g$ digits of size $\beta - 1$ are right-shifted beyond the rightmost guard digit register. Thus, for each q, Δy can be no larger than $\beta^{-t-g} - \beta^{-t-q}$. Since $\Delta d \geq \frac{1}{2}\beta^{-L-t}$, the guard digit registers must contain at least one nonzero digit. Consequently, $1 \leq q - g \leq t - 1$. (If $t = 1$, then case 4 with $q > g$ does not apply.) Moreover, since $0 \leq q \leq M_1$, we have $1 \leq g < q \leq \min(g + t - 1, M_1)$. In what follows, we make the reasonable assumption that $g + t - 1 \leq M_1$.

For each q ($1 \leq g < q \leq g + t - 1$), and each L ($0 \leq L \leq 1$), we conclude

$$\Delta y + (\Delta r - \Delta d) \leq \beta^{-t}(\beta^{-g} - \beta^{-q}) + \frac{1}{2}\beta^{-L-t} \qquad (2.5.3.19)$$

$$(\Delta d - \Delta y) \geq \frac{1}{2}\beta^{-L-t} - \beta^{-t}(\beta^{-g} - \beta^{-q}) \qquad (2.5.3.20)$$

Hence, for case 4 with $q > g$, $\beta \geq 2$, and $L = 0$,

$$E_{x\ominus y} = \frac{\Delta y + (\Delta r - \Delta d)}{(x \ominus y - \Delta r) + (\Delta d - \Delta y)}$$

$$\leq \frac{\beta^{-t}(\beta^{-g} - \beta^{-q}) + \frac{1}{2}\beta^{-t}}{\beta^{-1} + [\frac{1}{2}\beta^{-t} - \beta^{-t}(\beta^{-g} - \beta^{-q})]}$$

$$= \frac{(\beta^{-g} - \beta^{-q} + \frac{1}{2})}{(\beta^{t-1} + 1) - (\beta^{-g} - \beta^{-q} + \frac{1}{2})} \qquad (2.5.3.21)$$

$$= \frac{1}{[(\beta^{t-1} + 1)/(\beta^{-g} - \beta^{-q} + \frac{1}{2})] - 1} > 0 \qquad (2.5.3.22)$$

If, on the other hand, $L = 1$, then

$$E_{x\ominus y} = \frac{\Delta y + (\Delta r - \Delta d)}{(x \ominus y - \Delta r) + (\Delta d - \Delta y)}$$

$$\leq \frac{\beta^{-t}(\beta^{-g} - \beta^{-q}) + \frac{1}{2}\beta^{-t-1}}{(\beta^{-1} - \beta^{-3}) + [\frac{1}{2}\beta^{-t-1} - \beta^{-t}(\beta^{-g} - \beta^{-q})]}$$

$$= \frac{(\beta^{1-g} - \beta^{1-q} + \frac{1}{2})}{(\beta^{t} - \beta^{t-2} + 1) - (\beta^{1-g} - \beta^{1-q} + \frac{1}{2})}$$

$$= \frac{1}{[(\beta^{t} - \beta^{t-2} + 1)/(\beta^{1-g} - \beta^{1-q} + \frac{1}{2})] - 1} > 0 \qquad (2.5.3.23)$$

We show that for all q $(1 \leq g < q)$, (2.5.3.22) is strictly greater than (2.5.3.23). In fact, it is sufficient to demonstrate this for $q > g \geq 2$, since we have shown above that if $g = 1$ and $L \neq 0$, then case 4 does not apply.

Note that the desired result is obtained if we can show

$$\frac{\beta^{1-g} - \beta^{1-q} + \frac{1}{2}}{\beta^{-g} - \beta^{-q} + \frac{1}{2}} < \frac{\beta^{t} - \beta^{t-2} + 1}{\beta^{t-1} + 1} \qquad (2.5.3.24)$$

We see that

$$\frac{\beta^{1-g} - \beta^{1-q} + \frac{1}{2}}{\beta^{-g} - \beta^{-q} + \frac{1}{2}} = \beta - \frac{\frac{1}{2}(\beta - 1)}{\beta^{-g} - \beta^{-q} + \frac{1}{2}}$$

$$< \quad \beta - \frac{\frac{1}{2}(\beta - 1)}{\beta^{-2} + \frac{1}{2}}$$

$$= \frac{\beta^{-1} + \frac{1}{2}}{\beta^{-2} + \frac{1}{2}} \leq \alpha$$

where $\alpha = \frac{4}{3}$ if $\beta = 2$, and $\alpha = \frac{15}{11}$ if $\beta > 2$. Hence, (2.5.3.24) will be verified if we can show

$$\beta^{t} - \beta^{t-2} + 1 \geq \alpha(\beta^{t-1} + 1) \tag{2.5.3.25}$$

for each β and its corresponding α. Multiplying both sides of (2.5.3.25) by β^{-t} and rearranging, we obtain

$$1 - \beta^{-2} - \alpha\beta^{-1} \geq (\alpha - 1)\beta^{-2} \geq (\alpha - 1)\beta^{-t} \tag{2.5.3.26}$$

where the final inequality comes from the fact shown previously that $t \geq 2$ for case 4 to apply. From (2.5.3.26) we obtain immediately

$$\alpha^{-1} \geq \beta^{-1} + \beta^{-2} \tag{2.5.3.27}$$

Equality in (2.5.3.27) is achieved for ($\beta = 2$, $\alpha = \frac{4}{3}$), and strict inequality for ($\beta > 2$, $\alpha = \frac{15}{11}$). Thus, we have verified (2.5.3.24); and hence also that (2.5.3.22) is strictly greater than (2.5.3.23), which was desired.

From (2.5.3.19) and (2.5.3.20), observe that bound (2.5.3.22) was derived with $L = 0$ and $\Delta d = \frac{1}{2}\beta^{-t}$. But since $L = 0$, it follows that no post-adjustment shifts are performed, and the contents of the guard digits <u>before</u> the subtraction is also given by $\frac{1}{2}\beta^{-t}$. Note that this implies that the leftmost guard digit was nonzero before the subtraction. Hence, $1 \leq q \leq t$ in (2.5.3.22).

Observe that (2.5.3.21) is maximized when $q = t$. Consequently, for case 4 with $q > g$ and $\beta \geq 2$, we may conclude from (2.5.3.21) and (2.5.3.22),

$$E_{x \ominus y} \leq \frac{\frac{1}{2} + \beta^{-g} - \beta^{-t}}{(\beta^{t-1} + 1) - (\frac{1}{2} + \beta^{-g} - \beta^{-t})} \qquad (2.5.3.28)$$

$$< (\tfrac{1}{2} + \beta^{-g} - \beta^{-t})\,\beta^{1-t}$$

One can verify that (2.5.3.28) is indeed attained when

$$x = \beta^{-1} + \beta^{-t} \quad \text{and} \quad y = (\tfrac{1}{2} + \beta^{-g} - \beta^{-t})\,\beta^{-t}$$

The maximum relative error in $\oplus$ and $\ominus$ may now be summarized for scheme S_3 as follows.

Let

$$A_1 = \frac{(\frac{1}{2} - \beta^{-g})}{\beta^{t-1} + (\frac{1}{2} - \beta^{-g})} \qquad (2.5.3.29)$$

$$A_2 = \frac{(\frac{1}{2} - \beta^{-t})}{\beta^{t-1} + (\frac{1}{2} - \beta^{-t})} \qquad (2.5.3.30)$$

$$A_3 = \frac{(\frac{1}{2} - \beta^{-t-1})}{\beta^{t-1} + (\frac{1}{2} - \beta^{-t-1})} \qquad (2.5.3.31)$$

$$A_4 = \frac{\frac{1}{2}}{\beta^{t-1} + \frac{1}{2}} \qquad (2.5.3.32)$$

$$A_5 = \frac{\frac{1}{2} + (\beta^{-g} - \beta^{-t})}{\beta^{t-1} + \frac{1}{2} - (\beta^{-g} - \beta^{-t})} \qquad (2.5.3.33)$$

$$A_6 = \frac{\beta^{-g}(1 - \beta^{-t})}{\beta^{t-1} - \beta^{-g}(1 - \beta^{-t})} \qquad (2.5.3.34)$$

Given $1 \le g \le t$, it is easy to see that

$$A_1 \le A_2 < A_3 < A_4 \le A_5 \qquad (2.5.3.35)$$

Also, it can be shown that for all $1 \le g \le t$, we have $A_6 \le A_5$, with equality holding when $\beta = 2$, $g = 1$, and $t = 1$. Thus, for $\beta > 2$,

$$-A_2 \le E_{x\oplus y},\ E_{x\ominus y} \le A_5 \quad \text{if} \quad 1 \le g < t \qquad (2.5.3.36)$$

and for $\beta = 2$,

$$-A_3 \le E_{x\oplus y},\ E_{x\ominus y} \le A_5 \quad \text{if} \quad 1 \le g < t \qquad (2.5.3.37)$$

Note that bounds A_2 and A_3 are independent of g, the number of guard digits. Since the derivation of A_5 required $1 \le g < q \le t$, A_5 does not apply if $g \ge t$. In this case, the upper bounds in (2.5.3.36) and (2.5.3.37) are replaced by the next highest bound, A_4, which is seen to be independent of g. Thus, for $\beta > 2$,

$$-A_2 \le E_{x\oplus y},\ E_{x\ominus y} \le A_4 \quad \text{if} \quad g \ge t \qquad (2.5.3.38)$$

and for $\beta = 2$,

$$-A_3 \le E_{x\oplus y},\ E_{x\ominus y} \le A_4 \quad \text{if} \quad g \ge t \qquad (2.5.3.39)$$

When $1 \le g < t$, we see from (2.5.3.35) that $A_2 < A_3 < A_5$. Thus, for this case, the maximum relative error will be minimized by minimizing A_5. Since for any fixed t, A_5 is minimized by maximizing g, we conclude that <u>more</u> guard digits are better for scheme S_3 with $1 \le g < t$. Additional guard digits have <u>no effect</u> upon the maximum relative error in scheme S_2 if $g \ge t$.

2.5.4 Comparison Between Schemes S_1, S_2, and S_3

The worst-case relative error in each of the schemes is copied below for ease of reference.

$$S_1: \quad C = \frac{(1 - \beta^{-t})}{\beta^{t-1} + (1 - \beta^{-t})} \qquad \text{if } g \geq 1 \qquad (2.5.4.1)$$

$$S_2: \quad B_1 = \frac{\frac{1}{2}(1 + \beta^{-g})}{\beta^{t-1} + \frac{1}{2}(1 - \beta^{-g})} \qquad \begin{array}{l} \text{if } 1 \leq g < t \\ \text{or } g = t \;\&\; \beta = 2 \end{array} \qquad (2.5.4.2)$$

$$B_2 = \frac{\frac{1}{2}}{\beta^{t-1} + \frac{1}{2}} \qquad \begin{array}{l} \text{if } g > t \\ \text{or } g = t \;\&\; \beta \neq 2 \end{array} \qquad (2.5.4.3)$$

$$S_3: \quad A_5 = \frac{\frac{1}{2} + (\beta^{-g} - \beta^{-t})}{\beta^{t-1} + \frac{1}{2} - (\beta^{-g} - \beta^{-t})} \qquad \text{if } 1 \leq g < t \qquad (2.5.4.4)$$

$$A_4 = B_2 \qquad \text{if } g \geq t \qquad (2.5.4.5)$$

For purposes of comparison, we assume that each scheme employs at least one guard digit ($g \geq 1$), and that the fp numbers have fractions containing at least three digits ($t \geq 3$).

By comparing (2.5.4.1) - (2.5.4.5), we find that C exceeds the worst-case relative error for every case of scheme S_2, and for every case except one of scheme S_3. If $\beta = 2$ and $g = 1$, then C is strictly smaller than the maximum relative error in scheme S_3, given by A_5. Thus, when binary arithmetic is used with one guard digit, chopping is better than the S_3 scheme of rounding.

For $1 \leq g < t$, we find that $B_1 \leq A_5$, with equality holding only when $\beta = 2$ and $g = t - 1$. Thus, if less than an entire double precision accumulator is available, then rounding scheme S_2 produces a strictly smaller worst-case error than rounding scheme S_3, unless $\beta = 2$ and $g = t-1$, in which case, the worst-case errors for the two schemes are identical. (Rounding during pre-adjustment will not reduce the maximum relative error if $\beta = 2$ and $g = t-1$.)

If $\beta = 2$ and $g = t$, then the maximum relative error in S_2 is given by B_1, and the maximum relative error in S_3 is given by B_2. It is easy to show that $B_2 < B_1$. Thus, in this case, rounding in scheme S_3 produces _smaller_ worst-case relative errors than scheme S_2. (Rounding during pre-adjustment actually _increases_ the maximum relative error in this case, because the pre-adjustment round can cause an undesirable post-adjustment round.)

If $g \geq t$ and $\beta \neq 2$, or $g > t$ and $\beta = 2$, then the worst-case relative in S_2 is identical to the worst-case relative error in S_3. Thus, in this case, pre-adjustment rounding has _no effect_ upon the worst-case relative error. These observations are summarized in table 2.5.4.1.

	$g = 1$	$1 < g < t-1$	$g = t-1$	$g = t$	$g > t$
$\beta = 2$	$S_2 < S_1 < S_3$	$S_2 < S_3 < S_1$	$S_2 = S_3 < S_1$	$S_3 < S_2 < S_1$	$S_2 = S_3 < S_1$
$\beta > 2$	$S_2 < S_3 < S_1$	$S_2 < S_3 < S_1$	$S_2 < S_3 < S_1$	$S_2 = S_3 < S_1$	$S_2 = S_3 < S_1$

Table 2.5.4.1. Comparison of worst-case relative errors for addition schemes S_1, S_2, and S_3.

Taking into account the fact that a factor of $\log \beta_2 / \log \beta_1$ more bits are needed to represent a number in base β_1 than in base β_2, (2.5.4.1) -

(2.5.4.5) all indicate that base 2 would incur the smallest worst-case relative errors for any given number of bits, as we found for $\otimes$ and $\oslash$. Also, using these precise bounds, it is apparent that, contrary to Brent [4] and Cody [8], the systems ($\beta_1 = 2$, $t_1 = 22$) and ($\beta_2 = 4$, $t_2 = 11.5$) do <u>not</u> have identical worst case errors. (The latter scheme represents a 23-bit, base 4 number system.) Rather, (β_1, t_1) yields strictly tighter relative error bounds for each of S_1 - S_3 as well as for the multiplication and division schemes schemes presented in sections 2.5.3 and 2.5.4. (For multiplication and division, however, note that the derived bounds are not as tight as possible for all choices of β and t.) Two factors contribute to this result: First, comparing a base 2 representation with a base 4 representation having an extra bit of precision, the extra bit of precision is an extra bit of information that may be discarded when a negative error is incurred. Also, if the guard digits are chosen to hold $t_1 + 1$ bits, then the pre-adjustment round of scheme S_2 can induce an undesirable post-adjustment round with (β_2, t_2), but not with the (shorter) numbers of (β_1, t_1). Fortunately, it is true that for <u>fixed</u> β, the relative errors do indeed decrease with increasing t.

For completeness, we mention that Liddiard [49] has indicated a preference for base 2, because base 2 minimizes the variation in precision among the fp numbers, caused by a variable number of leading zero bits.

Though the worst-case relative error in scheme S_1 is independent of g, the number of guard digits, we saw in section 2.5.1 that the next largest relative error is smallest when g = 1. Examining (2.5.4.2) and (2.5.4.4), we see that if $1 \leq g < t$, then the worst-case relative error in both S_2 and S_3 is minimized by making g as <u>large</u> as possible. Note, however, that when

$g \geq t$ in scheme S_3, or $g > t$ in scheme S_2, additional guard digits have *no effect* upon the worst-case relative error. Let us consider what effect, if any, additional guard digits have upon the less-than-worst-case relative error when $g \geq t$.

Suppose that the exponents of x and y differ by $q \geq 0$. We saw in section 2.5.1 that if $x = \beta^{-1}$ and $q \geq t+2$, or if $x \neq \beta^{-1}$ and $q \geq t+1$, then digit position $t+1$ of the exact, normalized difference, $x-y$ will contain a digit of size $\beta - 1$. Thus, if sufficient guard digits are available to form $x-y$ exactly, then after rounding the result to t digits, we will find $x \ominus y = x$. Note also that if $q \geq t+1$, then digit position $t+1$ of the exact, normalized sum, $x+y$, will contain a digit equal to 0. Thus, if sufficient guard digits are available to form $x+y$ exactly, in this case also, rounding the result to t digits will yield $x \oplus y = x$.

Since $x+y$ and $x-y$ do not need to be formed explicitly when the exponents of x and y differ by $q \geq t+2$, it follows that maximum precision t-digit results may be obtained by forming $x+y$ or $x-y$ exactly using $t+1$ guard digits if $q \leq t+1$, and aborting the $\oplus$ or $\ominus$ if $q > t+1$. Note that no pre-adjustment rounding is required. Moreover, if $t+1$ guard digits are used, then it can be shown that S_2 and S_3 yield identical results, regardless of whether the $\oplus$ or $\ominus$ is aborted for $q > t+1$.

In summary, from the standpoint of accuracy and hardware cost, it appears so far that the choice of scheme S_3 using base 2 with $t+1$ guard digits for performing rounded addition is best, where the addition is aborted when $q > t+1$ for efficient operation. In fact, $t+1$ guard digits are not *necessary* to obtain maximum precision. In section 2.5.5 we introduce scheme S_5, which for any base uses only 1 guard digit and 2 guard bits to achieve the precision described here for that base with $t+1$ guard digits. Note that the maximum relative errors for scheme S_3 with $t+1$ guard digits are given by (2.5.3.38) and (2.5.3.39), and that these bounds are independent of $g \geq t$. Finally, note that if this scheme were to be unbiased, then the maximum positive relative error would be unchanged, but the magnitude of the negative relative error bound would increase to the same value as the positive relative error bound.

2.5.5 Efficient Schemes for Accurately Performing Rounded Addition: Schemes S_4 and S_5

Knuth [42, section 4.2.1, exercise 5] shows that any sum obtained by scheme S_3 with $t+1$ guard digits can be achieved with only 2 guard digits and the following rounding scheme.

Scheme S_4.

Let $\lfloor a \rfloor$ denote the greatest integer $\leq$ a (or floor(a)).

Let $\lceil a \rceil$ denote the least integer $\geq$ a (or ceiling(a)).

(1) Pre-Adjustment: y is right-shifted until its exponent equals the exponent of x. Let f_y denote the fractional part of y. For $x \oplus y$, replace f_y by

$$\beta^{-t-2} \lfloor \beta^{t+2} f_y \rfloor \qquad (2.5.5.1)$$

For $x \ominus y$, replace f_y by

$$\beta^{-t-2} \lceil \beta^{t+2} f_y \rceil \qquad (2.5.5.2)$$

(2) Addition: (Same as schemes S_2 and S_3)

(3) Post-Adjustment: (Same as schemes S_2 and S_3)

Note that both (2.5.5.1) and (2.5.5.2) imply that all digits of y right-shifted beyond the second guard digit register are discarded. However, in (2.5.5.2), the second guard digit is incremented if any of the discarded digits is nonzero.

Both Yohe [74] and Kulisch [46] have shown how the pre-adjustment increment required by (2.5.5.2) can be eliminated from scheme S_4 by using an additional 1 bit register. In their scheme, this "sticky" bit is set to 1 if any nonzero digit of y is right-shifted beyond the second guard digit register during pre-adjustment. Otherwise, the sticky bit remains 0. This sticky bit participates in $\oplus$ and $\ominus$ simply as an extra bit of precision for y. That is, it has no effect in $x \oplus y$, and in $x \ominus y$ its sole effect is to require a "borrow" from the least significant digit of x if any nonzero digits of y were discarded during pre-adjustment. It is straightforward to see that this scheme is equivalent to scheme S_4.

Thus far we have seen how a t-digit accumulator with 1 overflow bit, 2 guard digits, and a sticky bit may be used to form maximum precision sums without any pre-adjustment rounding. We now show how we may obtain the same precision, replacing the second guard digit by a single bit, and performing post-adjustment rounding with one shifting operation. (Garner [20] also mentioned that maximum precision sums were possible, with the second guard digit replaced by a single bit, but he did not give all the details for its implementation.) Later, we indicate how this scheme may be implemented so that it requires no more time than the slowest chopped addition of scheme S_1.

Scheme S_5.

We use the following notation:

OB ≡ overflow bit
AC ≡ t-digit accumulator
GD ≡ guard digit
GB ≡ guard bit
SB ≡ "sticky" bit
|| ≡ concatenation of bits

if $B_1 \rightarrow SL_1 \;\square\; B_2 \rightarrow SL_2 \;\square\; \cdots \;\square\; B_n \rightarrow SL_n$ **fi**

denotes a selection and execution of one of the statement lists $SL_1, \ldots, SL_n$, depending upon the Booleans $B_1, \ldots, B_n$.

The registers required for scheme S_5 are detailed below.

```
 _    ______________________________    _   _   _
|_|  |______________________________|  |_| |_| |_|
OB   <------------ AC ------------->   GD  GB  SB
```

Figure 2.5.5.1. Layout of registers for scheme S_5.

$\beta^{t+1} f_y - \lfloor \beta^{t+1} f_y \rfloor$	Prior to $x \oplus y$ or $x \ominus y$		After $x \oplus y$		After $x \ominus y$	
	GB	SB	GB	SB	GB	SB
= 0	0	0	0	0	0	0
> 0 & < 1/2	0	1	0	1	1	1
= 1/2	1	0	1	0	1	0
> 1/2	1	1	1	1	0	1

Table 2.5.5.1. Contents of GB and SB before and after forming $x \oplus y$ and $x \ominus y$.

(1) <u>Pre-Adjustment</u>: load y into AC; rightshift(AC || GD) until the exponent of y equals the exponent of x; set GB and SB as indicated by table 2.5.5.1.

(2) <u>Addition</u>: x ± (OB || AC || GD || GB || SB) is performed exactly, with GB and SB participating as two additional bits of precision for y.

(3) <u>Post-Adjustment</u>:

if OB = 1 → rightshift(OB || AC || GD) one digit position;
add β / 2 to (AC || GD)

```
□   OB = 0  &  leftmost digit of AC ≠ 0  →
                add β / 2 to (OB || AC || GD);

                if  OB = 1 →  rightshift( OB || AC ) one digit position
                □   OB = 0 →  skip
                fi

□   OB = 0  &  leftmost digit of AC = 0  →
                add 1 to (AC || GD || GB);
                leftshift( AC || GD ) until it is normalized
fi
```

Consider $x \oplus y$ formed by scheme S_5. Recall that we have assumed without loss of generality that x and y are normalized fp numbers with $x \geq y > 0$, and that the exponent of x equals 0. Thus, the exact sum, $x+y$, may be bounded as follows.

$$\beta^{-1} < x + y \leq (1 - \beta^{-t}) + (1 - \beta^{-t}) = 2 - 2\beta^{-t} \quad (2.5.5.3)$$

From (2.5.5.3) we see that $x+y$ can have a fraction-overflow value of at most 1. That is, after adding x and y, but before post-adjustment shifting and rounding, either OB = 0 and the leftmost digit of AC ≠ 0 (no fraction-overflow), or OB = 1 (fraction-overflow value = 1). In the former case, the first $t+1$ digits of $x+y$ are contained in (AC || GD), and a properly rounded t-digit sum is obtained by incrementing (OB || AC || GD) with $\beta/2$ and right-shifting (OB || AC) one digit position if fraction-overflow occurs. In the latter case, the first $t+1$ digits of $x+y$ are contained in (OB || AC), and a properly-rounded t-digit sum is obtained by right-shifting (OB || AC || GD) one digit position, and then incrementing (AC || GD) by $\beta/2$.

Note that in the latter case, the increment is equivalent to incrementing $x+y$ by $(\beta/2)\beta^{-t}$. We see from (2.5.5.3) that an increment of at least $2\beta^{-t}$ is necessary to produce a fraction-overflow value greater than 1, and that an increment must exceed $1 + 2\beta^{-t}$ to produce a fraction-overflow value greater than 2. Consequently, rounding $x+y$ as described above cannot yield a fraction-overflow value greater than 1 if $\beta < 4$, and the fraction-overflow value can never exceed 2 for any $\beta \geq 4$. Since the leftmost digit in AC can hold a "2" when $\beta \geq 4$, and since it can hold a "1" when $\beta < 4$, we conclude that fraction-overflow cannot occur from rounding a right-shifted sum.

Consider $x \ominus y$ formed by scheme S_5. Note that incrementing GD by $\beta/2$ will increase $x-y$ by $(\beta/2)\beta^{-t-1}$, and incrementing GB by 1 will increase $x-y$ by an amount not greater than β^{-t-1}. However, since

$$x - y < x \leq 1 - \beta^{-t}$$

an increment of at least β^{-t} is required to produce fraction-overflow. Thus, no fraction-overflow can occur from the S_5 scheme for $x \ominus y$.

If the exponents of x and y are equal, then prior to rounding, the exact difference, $x-y$, will be contained in AC, and (GD || GB || SB) will be 0. Thus, in this case, scheme S_5 cannot alter the exact difference contained in AC.

If the exponent of x exceeds the exponent of y by 1, then prior to rounding, the exact difference, $x-y$, will be contained in (AC || GD), and (GB || SB) will be 0. If the leftmost digit in AC is nonzero, then no left-shifting is required for post-adjustment, and a properly-rounded t-digit result will be placed in the accumulator by incrementing (OB || AC || GD) by $\beta/2$. On the other hand, suppose that the leftmost digit in AC is 0. Since GB is also 0, incrementing (AC || GD || GB) by 1 cannot alter the exact difference contained in (AC || GD). Subsequent left-shifting of (AC || GD) will place all nonzero digits of $x-y$ within AC.

Suppose that the exponent of x exceeds the exponent of y by 2 or more. If (GB || SB) is 0 prior to forming $x \ominus y$, then (GB || SB) has the desirable effect of not altering $x \ominus y$. If (GB || SB) is nonzero prior to forming $x \ominus y$, then some nonzero digits of y were shifted out of (AC || GD) during pre-adjustment. In this case, (GB || SB) necessitates a "borrow" from the least significant digit of x, as would have happened if no digits of y were lost. Finally, we see from table 2.5.5.1 that GB will contain a "1" after forming $x \ominus y$, only if

$$0 < \beta^{t+1} f_y - \lfloor \beta^{t+1} f_y \rfloor \leq \frac{1}{2} \qquad (2.5.5.4)$$

That is, GB will contain a "1" only if the digits of f_y shifted out of (AC || GD) during pre-adjustment are $\leq \frac{1}{2}\beta^{-t-1}$, but not all 0.

Note that when the exponent of x exceeds the exponent of y by 2 or more

$$x - y > \beta^{-1} - \beta^{-2} \geq \beta^{-2}$$

Consequently, $x - y$ can introduce <u>at most</u> one leading 0 into AC. If the leftmost digit in AC is nonzero, then as in the previous case, incrementing (OB || AC || GD) by $\beta/2$ will leave a properly-rounded t-digit result in AC. Otherwise, the first t nonzero digits of $x - y$ will be contained in (AC || GD), and (GB || SB) will be set in the manner indicated by table 2.5.5.1. Note that incrementing (AC || GD || GB) by 1 will cause a "carry" to propagate into GD only if GB contained a "1" prior to incrementing. From (2.5.5.4) we see that this occurs precisely when f_y contains nonzero digits of size $\leq (\beta/2)\beta^{-t-2}$. This condition correctly specifies when the guard digit should be incremented to yield a properly-rounded t-digit result in (AC || GD). Subsequent left-shifting can place this result entirely within AC. This completes our proof of the correctness of scheme S_5.

Now we indicate how rounded addition can be performed in essentially the same amount of time as chopped addition. This efficiency is possible for fp implementations that represent fp numbers in sign-magnitude form, but perform fp arithmetic in 2's complement. (Viz. IBM 360/370 [2,35,36] and DEC PDP-11 [57,58].)

In converting a negative 2's complement number to its corresponding sign-magnitude form, the 2's complement representation is usually complemented and incremented, so that its magnitude may be obtained. This action is performed regardless of whether chopped or rounded arithmetic is used. Thus, by combining the increment required for conversion with the increment required for rounding, a single addition may be used to perform both operations <u>simultaneously</u>. Hence, any rounded addition may be performed in essentially the same time as that required for a negative chopped sum. (A few extra hardware gate delays are needed to decide the increment size.)

Similar efficiency is attainable for fp implementations that perform arithmetic in 1's complement, regardless of the fp number representation. (In the CDC 6000/Cyber-70 series computers, 1's complement arithmetic is performed upon fp numbers with a 1's complement number representation [26,41].) To form a correct 1's complement sum, the carry-out from the sign-bit must be added to the least significant digit position in the accumulator. This action is performed regardless of whether chopped or rounded arithmetic is used. By combining the increment required to form a correct 1's complement sum with the increment required for rounding, a single addition may be used to perform both operations <u>simultaneously</u>, analogous to the case with 2's complement arithmetic. Once again, we see that rounded addition may be performed in essentially the same time as that required for chopped addition.

2.6 Summary of Floating-Point Hardware Design Considerations

We have analyzed the relative errors in various floating-point arithmetic schemes as a function of the base and number of guard digits employed. We obtained tight upper and lower bounds for the maximum relative errors in each scheme, and indicated how the schemes compare for each possible choice of base and number of guard digits.

One surprising result is that chopped addition yields smaller worst-case errors when _fewer_ guard digits are used, with one guard digit yielding the smallest worst-case errors. Computer designers can no longer justify attempts to avoid implementing rounded addition by adding more guard digits, since additional guard digits with chopping can only _increase_ the worst-case relative errors.

Another interesting result is that from the standpoint of minimizing worst-case relative errors, none of the schemes S_1, S_2, or S_3 is uniformly the best or worst for every possible choice of base, β, and number of guard digits, g. If $g = 1$ and $\beta = 2$, then the chopped addition scheme, S_1, is superior to rounding scheme S_3, but rounding scheme S_2 is superior to both. If $g = t$ and $\beta = 2$, then rounding scheme S_3 is best.

Maximum precision base β sums were shown to be attainable with scheme S_3 and $g = t+1$. Using scheme S_5, we showed how the hardware requirements for this precision can be reduced to $g = 1$ (one guard digit) and two additional bits. In this scheme, a second post-adjustment shift, required by conventional schemes to deal with fraction-overflow caused by rounding, is never needed.

Using our tight error bounds, we confirmed the commonly held opinion that arithmetic in base 2 yields the smallest worst-case relative errors [4,5,8,43]. However, from these bounds it is apparent that, contrary to Brent [4] and Cody [8], base 4 yields greater worst-case errors than base 2, even if the base 4 numbers are granted one more bit of precision than the base 2 numbers. Also, we mentioned that Liddiard [49] had indicated a preference for base 2, because base 2 minimizes the variation in precision among the floating-point numbers, caused by a variable number of leading zero bits.

Finally, we have indicated how certain implementations of floating-point addition can produce properly-rounded sums in essentially the same amount of time as they take to produce chopped sums. Thus, for these implementations, we argue that rounded addition is <u>not</u> inferior to chopped addition on the grounds of efficiency, as is commonly assumed.

3. Program Correctness Proofs

3.1 Overview

In section 3, we introduce Dijkstra's Guarded Command Language and present two approaches for reasoning about floating-point programs written in this language.

The first approach associates a set of floating-point numbers with each floating-point expression and models the assignment statement as a nondeterministic selector of one of the elements in the set.

The second approach models the floating-point operations by single-valued functions, whose significant properties are characterized by a small set of axioms. Interesting properties of floating-point arithmetic are derived from the axioms, and examples are given to illustrate how these axioms and theorems may be used to prove floating-point programs correct.

The common practice of modelling the floating-point operations by a single function that crops the result of the corresponding exact operation is shown to be invalid for many implementations of floating-point arithmetic.

3.2 Background and Motivation

> Be sure of it; give me the ocular proof.
> -- Shakespeare, Othello

> Let us first of all follow reason, it is the surest guide. It warns us itself of its feebleness and informs us of its own limitations.
> -- Anatole France, Credo of a Sceptic

"... A proof is an argument that convinces the reader (or listener) of the truth of a statement" [25]. The ability of a proof to convince depends upon the clarity of its presentation, the comprehensiveness of its argument, and whether or not the inferences used can be considered self-evident. The assumptions stated during the course of the argument serve to define and delineate the object of the proof. Thus, from this perspective, the assumptions, if stated, have no bearing upon the convincing power of the proof, but _only_ upon the proof's domain of applicability.

Proofs may be more or less _formal_ depending upon the extent to which mathematics is used to assist the argument. In particular, mathematics is well-suited for making the argument precise, explicit, and well-organized. Unfortunately, the amount of detail required to make an argument precise and explicit sometimes becomes excessive, resulting in a proof that is cumbersome to generate and difficult to understand, thus detracting from the goal of a clear presentation. In such cases, an argument with fewer explicit details is warranted, leading to a proof that is more _informal_. The convincing power of an informal proof, however, depends upon its ability to evoke the belief that any details not given precisely and explicitly could be supplied upon request.

In the preceding discussion, we have used the words "belief", "clarity", "self-evident", "can be considered", and "ability to evoke". These words suggest that proofs possess certain subjective qualities. De Millo, Lipton, and Perlis [13] build a good case for this point of view, emphasizing that a proof must be confirmed by an educated community before it "can be considered" valid. The proof may fail to be confirmed because its presentation is obscure, its argument is incomplete, or there is evidence that an inference is incorrect. In the latter case, the proof is incorrect, and in any of these cases, the proof is inadequately unconvincing. (Proving a computer program to be correct by executing it on test cases often fails to convince because of argument incompleteness.)

By noticing that proofs are somewhat subjective, this does not imply that the truths they purport to demonstrate must necessarily also be subjective. On the contrary, we insist that such truths are entirely objective: A given algorithm either meets its specification or it doesn't. An algorithm either implements a specified function or it doesn't. It either runs in $N \log N$ time or it doesn't. Thus, if a proof is confirmed by a number of informed judges, we have no _guarantee_ that the proof is correct. However, our confidence in the proof, and hence also the consequent of the proof, is dramatically increased.

Although confidence in a proof builds after it is confirmed by an informed community, proofs do have value even in relative isolation. Personal misconceptions parading under the guise of "obvious truths" are often dispelled by a considered attempt to explain why these "truths" are so. And, if there is only one person to be brought to a particular way of thinking, is there a more satisfactory way than with a carefully planned, rational argument?

Let us consider program correctness proofs. To say that a program has been proven "correct" means that the program has been shown to satisfy a particular specification. Indeed, _apart from its specification, the "correctness" of a program is a meaningless concept_.

Hull [34] has expressed confusion about this point, wondering what it means to claim that a particular piece of numerical software is "correct". For a given problem, some programs are faster than others, some are more accurate than others, and some are reliable for a wider range of data than others. Also, he asks, "What is meant by the correctness of a program for playing chess?" The solution to this dilemma is actually quite simple. The specification for a program, by definition, is a statement of all the necessary properties of the program. To say that the program has been proven "correct", therefore, means that it meets <u>all</u> of these properties. Attainment of additional properties, such as the <u>quality</u> of a program, in so far as they are unspecified, is a concern separate from that of correctness. (Compare elegant vs. baroque proofs in mathematics.) An appropriate question to ask, however, is, "What properties <u>should</u> be required of a particular program?" This question can only be answered in view of the program's intended use.

Hoare [29] has observed that correctness proofs are useful not only to argue for the reliability of computer programs, but also as documentation and as aids to program portability and modifiability. The assumptions stated during the course of a proof serve to define and delineate the domain for which the reliability of a program segment is assured, and indicates why.

An individual who claims a distaste for these proofs or doubts their usefulness on the grounds that they are "long, ugly, and boring" [14], or difficult to generate and difficult to understand, has misunderstood what a proof is. We reiterate, a proof is nothing more or less than a convincing argument. As long as the trustworthiness of computer programs is of interest, we have no recourse but to rely upon correctness arguments (or proofs). In fact, it may be argued that even if test cases are used to instill confidence in a particular program, reasons should be given to explain clearly what the tests demonstrate and why anyone's level of confidence should be raised.

If, for whatever reason, one chooses to produce an inconclusive argument for the correctness of his program, it is imperative that he be cognizant of the following: First, that his argument is not conclusive; second, that techniques are now available to produce a conclusive argument; and finally, that choosing the bumpy road to hit-or-miss reliability may have disasterous consequences when a "miss" becomes a "hit" during a production run.

Distaste for the formalism of some program correctness proofs is not a valid criticism of all correctness proofs, but only of certain presentations. However, this distaste suggests that for human readability, correctness proofs should be presented informally, introducing formalism only where it is essential for clarity and precision.

Proofs that are substantially formal, however, have value for three reasons. First, they make the analysis explicit so that errors become apparent. Second, they systematize the reasoning so that mechanical assistance becomes possible. And finally, because they are explicit and systematic, they result in a better understanding of the reasoning involved in informal proofs.

Mathematical formalism has been used to reason about the correctness of computer programs since the ENIAC. Von Neuman and Goldstine [68], in 1947, and Turing [65], in 1948, published papers analyzing the cropping errors in matrix computations. These papers are said to have "laid the foundation for modern error analysis" [70]. In fact, associating logical assertions with computer programs also appears to have its origin with von Neuman, Goldstine, and Turing [22,66].

The papers by Floyd [18], in 1967, and Hoare [29], in 1969, urged that program semantics be placed on a solid axiomatic foundation to make the benefits of formal deductive logic accessible to program correctness proofs. Hoare observed, "[for any deterministic computer]...all the properties of a program and all the consequences of executing it in any given environment can, in principle, be found out from the text of the program itself by means of purely deductive reasoning." Floyd and Hoare are considered by some to have done for program correctness what Euclid did for geometry [47].

The first proofs that made use of explicit axioms and formal deductive logic treated small, deterministic, sequential programs that use exact arithmetic [30]. Subsequently, formal proof techniques were developed to reason about procedures with parameters and recursion [31], nondeterminism [15,16], concurrency [56], and security [69]. These techniques have been applied sucessfully to large programs [24,69], but in each case, exact arithmetic was assumed.

Since the time of the ENIAC, numerical analysts have constructed numerous proofs of bounds on the acumulated error and termination (or convergence) for programs that use (inexact) floating-point arithmetic. However, with only a few exceptions [12,37,60], the proofs of error bounds assume termination, and the proofs of termination assume exact arithmetic. Moreover, the techniques that have been applied seem quite heuristic, relying upon the exceptional skill and experience of the numerical analyst. In fact, it is precisely because of the exceptional skill required to write floating-point programs that the term "analyst" has been applied to those individuals who do.

Various models have been proposed for studying floating-point arithmetic. Some of the best known are the "backward" or "a priori" error model of Wilkinson [70,71], the "interval" error model of Moore [53,54], and the "significance" model of Metropolis [50,51]. More recently, alternative models have been proposed by Miller et. al. [38,52], Brown [6], Oliver [55], and Aggarwal et. al. [1].

The analyses that employ these models have, to the best of this author's knowledge, all side-stepped the problem of rigorously proving certain relationships among program variables to be invariant, in spite of inexact arithmetic. Some analyses include a detailed account of the error incurred in each floating-point expression, but only on a single pass through a loop [60]. Other analyses apply only after the loops in these programs are "unwound" a fixed number of times and replaced by straightline code [38,52]. Still others claim certain invariant relationships, but neglect to rigorously prove that these relationships are, indeed, invariant [11,32,37,70]. Moreover, it appears that systematic techniques necessary to do this have, prior to this thesis, not been developed.

The work presented in section 3 illustrates various approaches for wedding Wilkinson's "backward" error model of floating-point arithmetic with Dijkstra's calculus of "weakest pre-conditions" [16] in an attempt to place the analysis of floating-point programs on a solid axiomatic foundation.

3.3 Dijkstra's Guarded Command Language (GCL)

Because of the expressive power of Dijkstra's Guarded Command Language (GCL) [16], we shall use it to describe our programs. This presents no difficulty for applying the techniques of section 3 to programs expressed in PASCAL, ALGOL, PL/I, or FORTRAN, since (except for PL/I's **fork** and **join**) the control structures of these languages can be expressed in terms of GCL.

GCL contains five primitive statement types and a construct for combining them:

1. [empty statement]	**skip**
2. [halt statement]	**abort**
3. [assignment statement]	x := E
4. [alternative statement]	**if** $B_1 \rightarrow SL_1$ $\square$ $\cdots$ $\square$ $B_n \rightarrow SL_n$ **fi**
5. [iterative statement]	**do** $B_1 \rightarrow SL_1$ $\square$ $\cdots$ $\square$ $B_n \rightarrow SL_n$ **od**
6. [composition of statements]	S_1; S_2; $\cdots$; S_n

where "E" denotes an arithmetic expression, "B_i" denotes a Boolean expression, "S_i" denotes one of the statements (1) - (5), and "SL_i" denotes a list of one or more statements, combined as in (6). In the alternative and iterative statements, Boolean B_i is said to be a <u>guard</u> for statement list SL_i, because SL_i can only be executed when Boolean B_i is true. Indeed, GCL gets its name from the fact that it employs the guarded command constructs, "$B_i \rightarrow SL_i$". (We say more about each of the statement types below.) The syntax of E and B_i shall be identical to that of PASCAL. We refer the interested reader to either of the references [72,73] for a precise definition.

The assignment statement is similar to that in PASCAL, ALGOL, PL/I, and FORTRAN, except for two features. First, expression evaluation is not permitted to have side-effects. That is, evaluation of E is not allowed to change the value of any variable. And secondly, the assignment statement is extended to allow several distinct simple variables on the left-hand side and several expressions on the right-hand side. For example,

$$x_1, x_2, x_3 := E_1, E_2, E_3$$

The above statement is executed as follows. First, the values denoted by E_1, E_2, and E_3 are determined. Then the values are assigned to the simple variables x_1, x_2, and x_3, respectively. This statement is particularly helpful for interchanging the values of two variables:

$$x, y := y, x$$

For the alternative statement, the guards $B_1, B_2, \cdots, B_n$ are evaluated in some order. If guard B_i is found to be true, then the corresponding statement list SL_i is executed. If two or more guards are true simultaneously, then a nondeterministic (arbitrary) choice between the true guards is made, and the corresponding statement list is executed. If none of the guards is found to be true, then the program aborts.

Because of the ability of the alternative statement to nondeterministically select from a number of alternatives, this statement allows an algorithm to be expressed in its most general form, freeing its definition from artificially biased choices. For example,

$$\begin{array}{ll} \textbf{if} & x \geq y \rightarrow z := x \\ \square & y \geq x \rightarrow z := y \\ \textbf{fi} & \end{array}$$

is a completely general description of an algorithm for setting z to the maximum of x and y. Any deterministic implementation of this algorithm is a special case of the description, biased by the treatment of the case $x = y$.

As a final consideration for the alternative statement, we note that since the program aborts when all the guards are found to be false, this statement encourages the programmer to consider every possible alternative. The mathematical notions of functional definition by cases and proof by cases are natural analogs to this statement.

The iterative statement (or "loop") is similar to the alternative statement: The guards $B_1, B_2, \cdots, B_n$ are evaluated in some order, and if guard B_i is found to be true, the corresponding statement list SL_i is executed. If two or more guards are true simultaneously, then a nondeterministic choice between the true guards is made, and the corresponding statement list is executed. The activity of evaluating guards, selecting and executing statement lists is continued until all the guards become false. If it is always the case that some guard is true, this activity continues forever; hence, we have an "infinite loop". If all the guards are initially false, none of the statement lists is executed, and the effect of the iterative statement is equivalent to that of a **skip**.

3.4 Predicates and Predicate Transformations

A *predicate* is a statement about certain variables that may be true or false, depending on the possible values of those variables. For example,

$$x \leq y$$

is a predicate, which is true if and only if all the possible values of x are ≤ each possible value of y. We shall use predicates, such as the one above, to describe the state of program variables.

Let "wp(M, P)" denote the set of all initial states such that activation of mechanism M is guaranteed to result in an activity that terminates, achieving a final state satisfying predicate P. For example,

$$wp(\text{ "}x := x+1\text{"},\ x \geq 0\ )$$

denotes the set of all possible initial values of x such that executing $x := x+1$ is guaranteed to terminate with $x \geq 0$. Or, expressed simply, $x \geq -1$.

"wp(M, P)" is called the *weakest pre-condition* for mechanism M and *post-condition* P, because it denotes the *largest* set of states for which P is attained after executing M. Thus, "wp(M, P)" gives the necessary and sufficient condition for P to be true after executing M.

wp is known as a *predicate transformer*, because given M, wp maps any predicate P, specifying a post-condition, into another predicate, specifying the corresponding pre-condition. In the above example, wp mapped "$x \geq 0$" to "$x \geq -1$".

In the next section we define wp for each of the GCL statements, and indicate how they can be used to reason about the correctness of programs expressed in GCL. However, before leaving the present section, we describe four important properties of wp, due to Dijkstra [16].

1. [Law of Excluded Miracle] For any mechanism M, we have

$$wp(\ M,\ false\ )\ =\ false$$

No states satisfy the predicate "false". Thus, the Law of the Excluded Miracle says that for any mechanism M, no initial state can cause M to terminate without reaching a final state. For mechanisms M that never terminate, there is no initial state for which any predicate P is ultimately attained. Thus, if M runs indefinitely, then for <u>all</u> P, wp(M, P) = false.

2. [Law of Monotonicity] For any mechanism M and any post-conditions Q and R such that

$$Q \Rightarrow R \quad \text{for all states,}$$

we also have

$$wp(\ M,\ Q\ ) \Rightarrow wp(\ M,\ R\ )$$

This law is obvious, since $Q \Rightarrow R$ means that the set of states denoted by Q is a subset of the set of states denoted by R.

3. [Law of Conjunction] For any mechanism M and any post-conditions Q and R, we have

$$[\ wp(\ M,\ Q\ )\ \&\ wp(\ M,\ R\ )\ ]\ =\ wp(\ M,\ Q\,\&\,R\ )$$

This law states that for any mechanism M, activating M in an initial state is guaranteed to yield a final state satisfying the conjunction of Q and R, if and only if that initial state is guaranteed to yield a final state satisfying Q <u>and</u> satisfying R.

4. [Law of Disjunction] For any mechanism M and any post-conditions Q and R, we have

$$[\ wp(\ M,\ Q\ )\ \vee\ wp(\ M,\ R\ )\ ]\ \Rightarrow\ wp(\ M,\ Q \vee R\ )$$

This law states that for any mechanism M, activating M in an initial state is guaranteed to yield a final state satisfying the disjunction of Q and R, provided that the initial state is guaranteed to yield a final state satisfying Q <u>or</u> satisfying R. If mechanism M is deterministic, then the implication in the other direction also holds. For nondeterministic M, however, the reverse implication is not valid. We refer the interested reader to the reference [16] for an explanation.

3.5 <u>Semantic Characterization of GCL</u>

Using the wp predicate transformer, the semantics of the GCL primitives is precisely defined here in terms of their effect upon predicates. Except for theorem 1, the treatment in this section is due to Dijkstra [16].

Axiom A1. [Empty Statement] For any post-condition R,

wp("**skip**", R) = R

That is, **skip** does not alter the state of any program variables.

Axiom A2. [Halt Statement] For any post-condition R,

wp("**abort**", R) = false

That is, there is no initial state in which **abort** may be activated to yield a properly terminating activity.

Axiom A3. [Assignment Statement] For any arithmetic expression E and any post-condition R,

$$wp(\text{"}x := E\text{"},\ R\ ) \ =\ \mathbf{Dom}(\ E\ )\ \&\ R^{x}_{E}$$

where **Dom**(E) is a predicate representing the set of states for which E is defined, and R^{x}_{E} means that all free occurrences of x in R are replaced by E. Thus, x := E is guaranteed to terminate with post-condition R valid, if both **Dom**(E) and R^{x}_{E} are valid before executing x := E. (We refer the reader to section 3.4 for an example.) In case := has several left and right-hand sides, the definition is similar. For example,

$$wp(\text{"}x_1,\ x_2,\ x_3 := E_1,\ E_2,\ E_3\text{"},\ R\ ) \ =$$
$$\mathbf{Dom}(\ E_1)\ \&\ \mathbf{Dom}(\ E_2)\ \&\ \mathbf{Dom}(\ E_3)\ \&\ R^{x_1,x_2,x_3}_{E_1,E_2,E_3}$$

where $R^{x_1,x_2,x_3}_{E_1,E_2,E_3}$ means that all free occurrences of x_1, x_2, and x_3 in R are simultaneously replaced by E_1, E_2, and E_3, respectively.

Axiom A4. [Alternative Statement] For any set of Booleans $B_1, \cdots,$ B_n, and any post-condition R,

$$wp(\text{"}\mathbf{if}\ B_1 \rightarrow SL_1\ \Box \cdots \Box\ B_n \rightarrow SL_n\ \mathbf{fi}\text{"},\ R\ )$$
$$=\ (\ B_1 \vee \cdots \vee B_n\ )$$
$$\&\ (\ \forall i :\ 1 \le i \le n :\ B_i \ \Rightarrow\ wp(\ SL_i,\ R\ )\ )$$

That is, **if** ··· **fi** is guaranteed to terminate with post-condition R valid, if and only if before executing **if** ··· **fi**, one of the B_i is true, **and** for any true B_i, executing the corresponding statement list, SL_i, is guaranteed to terminate with R valid.

[Iterative Statement or "Loop"] Instead of defining wp for the loop, we introduce a theorem which may be proven from the definition, and which for our purposes, we shall find more useful. The interested reader is refered to [16] for the definition, and for the proof of a similar theorem.

Before stating the theorem, we introduce the following notation. We shall call an arbitrary set A <u>discrete</u> if and only if for any x and y in A there are only a finite number of distinct values z in A satisfying $x \le z \le y$. For example, any finite set is discrete; also, the integers form a discrete set. Let $\mathbb{R}$ denote the set of real numbers.

<u>Theorem 1</u>. If for some set of Booleans $B_1, \cdots, B_n$, and some predicate P,

(1) $[\ \forall i : 1 \le i \le n : P \ \&\ B_i \Rightarrow wp(\ SL_i, P\ )\ ]$

for some t defined on a discrete subset of $\mathbb{R}$, and some T not appearing in any SL_i,

(2) $[\ \forall i : 1 \le i \le n : P \ \&\ B_i \Rightarrow t \ge 0\ ]$

(3) $[\ \forall i : 1 \le i \le n : P \ \&\ B_i \Rightarrow wp(\ \text{"}T := t;\ SL_i\text{"},\ t < T\ )\ ]$

then

$$P \Rightarrow wp(\ \text{"}\mathbf{do}\ B_1 \rightarrow SL_1 \ \Box \cdots \Box\ B_n \rightarrow SL_n\ \mathbf{od}\text{"},\quad P\ \&\ \neg(\ B_1 \vee \cdots \vee B_n\ )\ )$$

Proviso (1) states that P is <u>invariant</u>. That is, P is guaranteed to be true after executing any SL_i, if P and the corresponding B_i is true beforehand.

Provisos (2) and (3) are conditions sufficient to ensure that the loop will terminate. Specifically, proviso (2) states that if some SL_i can be executed (because B_i is true), then t must not be less than 0. Proviso (3) states that for each SL_i that can be executed, executing SL_i is guaranteed to decrease t by some positive amount. Thus, since t is bounded below and monotone decreasing on a discrete set, the loop is guaranteed to terminate. In fact, t is a bound on the number of iterations that can be performed.

Finally, given provisos (1) - (3), the consequent of the theorems is that the loop is <u>guaranteed</u> to terminate with P true and $B_1, \ldots, B_n$ false, if P is true before execution.

Axiom A5. [Composition of Statements] For any post-condition R,

$$wp(\text{"}S_1;\ S_2\text{"},\ R\ )\ =\ wp(\ S_1,\ \ wp(\ S_2,\ R\ )\ )$$

That is, post-condition R is guaranteed to hold after executing S_1; S_2, if and only if S_1 is guaranteed to terminate in a state satisfying wp(S_2, R). The generalization of wp for the composition of an arbitrary number of statements is straightforward and left to the reader.

Let us examine the tools we have accumulated thus far. We have the GCL language for writing programs; we can use predicates to specify the set of states that are acceptable when the program terminates; and, we can use the wp transformer on the text of the program to mechanically derive all the initial states that are guaranteed to yield one of the desired final states. Consequently, a program can be considered "correct" for any specified set of initial states contained by the derived set. We illustrate this approach in the next few sections.

3.6 A First Approach for Proving Program Correctness

Come now, and let us reason together. -- Isaiah 1:17

Agree with me if I seem to speak the truth. -- Socrates

Given a hardware or software implementation of some scheme of fp (floating-point) arithmetic, one could, in principle, define a function that modeled each fp operation *exactly*. However, because of the mathematical tedium that this approach would incur, it is doubtful whether any insight beneficial to proving programs correct would be obtained by doing so.

The approach adopted here and in section 3.7 is to model the fp operations by a small set of axioms, which are intended to capture the fundamental properties in common with many fp arithmetic schemes. Because of this approach, irrelevant implementation details are absent from our proofs. Also, each result proved is ensured a broad domain of applicability.

As a first approach to reasoning about fp programs, let us associate a set of fp numbers with each fp expression. Recall that $\mathbb{F}$ denotes the set of all fp numbers supported by some fp implementation. Now, let

$$\circledcirc \equiv \text{one of the fp operators } \oplus, \ominus, \otimes, \oslash$$

$$\cdot \equiv \text{the exact operator corresponding to } \circledcirc$$

$$x \equiv \text{an element of } \mathbb{F}$$

$$E_1, E_2 \equiv \text{two fp expressions}$$

then we define

$$\overline{x} \equiv \{ x \}$$

$$\overline{-x} \equiv \{ -x \}$$

$$\overline{E_1 \circledast E_2} = \{ (E_1 \cdot E_2)(1 + \delta) \in \mathbb{F} : -\lambda \le \delta \le \mu \}$$

where λ and μ bound the relative error in all the operators $\oplus$, $\ominus$, $\odot$, $\oslash$. (Typically, $0 \le \lambda, \mu << 1$.) Thus, for w, x, y, z in $\mathbb{F}$,

$$\overline{x \ominus y} = \{ (x - y)(1 + \delta_1) \in \mathbb{F} : -\lambda \le \delta_1 \le \mu \}$$

$$\overline{w \oslash z} = \{ (w / z)(1 + \delta_2) \in \mathbb{F} : -\lambda \le \delta_2 \le \mu \}$$

$$\overline{w \oslash (x \ominus y)} = \{ (w / z)(1 + \delta_2) \in \mathbb{F} : z \in \overline{x \ominus y} \ \& \ -\lambda \le \delta_2 \le \mu \}$$

Note that if $x - y = 0$, then $\overline{x \ominus y} = \{ 0 \}$ and $\overline{w \oslash (x \ominus y)} = \{ \}$. Moreover, if $\lambda = \mu = 0$, then the fp operators are exact, and $\overline{E_1 \circledast E_2} = \{ E_1 \cdot E_2 \}$.

Now, for any set $\overline{E} = \{ v_1, \cdots, v_n \}$, let us interpret the assignment statement "x := E" by

$$\begin{array}{lll} \textbf{if} & \text{true} \rightarrow & x := v_1 \\ \Box & \text{true} \rightarrow & x := v_2 \\ & \ \vdots & \\ \Box & \text{true} \rightarrow & x := v_n \\ \textbf{fi} & & \end{array}$$

That is, "x := E" denotes a nondeterministic selection of one of the members of $\overline{E}$.

From the definition of wp given in section 3.5, we conclude

$$\begin{aligned} &wp(\text{"}x := E\text{"}, P) = \\ &\qquad \text{Dom}(E) \ \& \ wp(\text{"}x := v_1\text{"}, P) \\ &\qquad\qquad \& \ wp(\text{"}x := v_2\text{"}, P) \\ &\qquad\qquad\quad \cdot \end{aligned}$$

$$\vdots$$

$$\& \ \ wp(\text{"}x := v_n\text{"},\ P)$$

where Dom(E) is a predicate characterizing the set of states for which E is defined.

This approach leads us to the following fp axiom, which replaces the assignment statement axiom of section 3.5.

Axiom A3'. For any fp expression E and any post-condition R,

$$wp(\text{"}x := E\text{"}, R) = \mathrm{Dom}(E) \ \& \ (\forall v \in \bar{E} : R_v^x)$$

That is, x := E is guaranteed to terminate with post-condition R valid, if before executing x := E, it is the case that E can be evaluated in a state for which E is defined, <u>and</u> R_v^x is valid for every possible value of E. Note that axiom A3' reduces to axiom A3 when $\lambda = \mu = 0$.

We illustrate the use of axiom A3' in the following correctness proof.

<u>Example</u>: Given $n \geq 0$ and fp vectors x[1:n] and y[1:n]. Use fp arithmetic to determine the <u>exact</u> inner product of $\bar{x}$[1:n] and $\bar{y}$[1:n], where $\bar{x}$ and $\bar{y}$ are perturbed versions of x and y. Bound the amount by which $\bar{x}$ and $\bar{y}$ deviate from x and y.

<u>Program</u>:

```
z, k := 0, 0;  {P}
do  k ≠ n  →  {P  &  k ≠ n}
              k := k + 1;
              z := z ⊕ x[k] ⊛ y[k]   {P}
od
```

Note that k is incremented exactly. To simplify the presentation, we assume k is always in range, and that neither overflow nor underflow occurs. "{" and "}" delimit predicates, called assertions, which are claimed to be valid whenever the flow of control reaches the points at which they occur in the given program.

Predicate P is defined by

$$P \equiv [\ 0 \le k \le n \ \ \& \ \ z = z^{(k)} \ \ \& \ \ z^{(0)} = 0 \ \ \&$$

$$(\forall j : 1 \le j \le k : \exists \delta_1^{(j)}, \delta_2^{(j)} : \ -\lambda \le \delta_1^{(j)}, \delta_2^{(j)} \le \mu \ \ \&$$

$$z^{(j)} = (\ z^{(j-1)} + x[j]\, y[j]\, (1 + \delta_1^{(j)}))\, (1 + \delta_2^{(j)})\ )\ ]$$

P gives bounds on k and states that variable z is the kth element of some sequence $\{ z^{(j)} \}_{j=0}^{k}$, where $z^{(j)}$ is defined in terms of $z^{(j-1)}$, the exact product x[j] y[j], and certain perturbations. Note that P is claimed to be true regardless of the number of times the **do** ··· **od** loop is executed. Thus, in this sense, P is claimed to be invariant.

We now justify our claim that predicate P is invariant.

Proof of the Invariance of P:

It is trivial to see that P holds immediately before executing the **do** ··· **od** loop, and that $k \ne n$ is valid if the loop guard is found to be true.

If P is to hold at the end of the loop, for each iteration, then by axiom A5, the following must hold at the beginning of the loop, for each iteration.

$$Q \equiv wp(\text{"}k := k + 1\text{"},\ R)$$

where $$R \equiv wp(\text{"}z := z \oplus x[k] \circledast y[k]\text{"},\ P)$$

Since, by assumption, overflow and underflow do not occur, axiom A3' applies to R with

$$\mathbf{D}\mathrm{om}(\ z \oplus x[k] \circledast y[k]\ ) \ = \ \text{true}$$

Thus,

$$R \ = \ (\ \forall v \in \overline{z \oplus x[k] \circledast y[k]} \ : \ P_v^z\ )$$

Also, by assumption, k is incremented exactly and is in range. Consequently, axiom A3 applies to Q with

$$\mathbf{D}\mathrm{om}(\ k + 1\ ) \ = \ \text{true}$$

Thus,

$$\begin{aligned} Q \ &= \ R^k_{k+1} \\ &= \ (\ \forall v \in \overline{z \oplus x[k+1] \circledast y[k+1]} \ : \ P^{z,\ k}_{v,\ k+1}\ ) \end{aligned}$$

Therefore, by performing the indicated substitutions in Q and applying theorem 1, we see that the invariance of P is established if

$$P \ \& \ k \neq n \ \Rightarrow \ Q$$

where

$$\begin{aligned} Q \ = \ (\ \forall v \in \overline{z \oplus x[k+1] \circledast y[k+1]} \ : \\ &[\ 0 \le k+1 \le n \ \& \ v = z^{(k+1)} \ \& \ z^{(0)} = 0 \ \& \\ &(\ \forall j \ : \ 1 \le j \le k+1 \ : \ \exists \delta_1^{(j)},\ \delta_2^{(j)} \ : \\ &-\lambda \le \delta_1^{(j)},\ \delta_2^{(j)} \le \mu \ \& \\ &z^{(j)} = (\ z^{(j-1)} + x[j]\,y[j]\,(1 + \delta_1^{(j)}))\,(1 + \delta_2^{(j)})\ )\ ]\ ) \end{aligned}$$

Note that

$$P \ \& \ k \neq n \ \Rightarrow \ 0 \leq k < n$$
$$\Rightarrow \ 0 \leq k+1 \leq n$$

From the definition of $\bar{E} = \overline{z^{(k)} \oplus x[k+1] \otimes y[k+1]}$ we have

$$(\forall v \in \bar{E} : \ (\exists \delta_1, \delta_2 :$$
$$-\lambda \leq \delta_1, \delta_2 \leq \mu \ \&$$
$$v = (z^{(k)} + x[k+1]\, y[k+1]\, (1 + \delta_1))\, (1 + \delta_2) \ ) \ )$$

Since P implies the existence of the sequences $\{\delta_1^{(j)}\}_{j=1}^{k}$, $\{\delta_2^{(j)}\}_{j=1}^{k}$, $\{z^{(j)}\}_{j=0}^{k}$, with $z^{(k)} = z$, the desired result, namely, $P \ \& \ k \neq n \Rightarrow Q$, follows.

Therefore, P is invariant with respect to the loop in the given program. Also, since k is incremented by 1 each iteration using exact arithmetic, the program is guaranteed to terminate attaining the assertion

$$P \ \& \ k = n \qquad \text{QED}$$

Let us summarize what we have accomplished thus far. By specifying P, we captured in a static, mathematical object what we thought to be the essence of a particular, sequential program. Then, using purely mechanical means, we verified that the state of affairs described by $P \ \& \ k = n$ would indeed be established by the program when it terminates. To complete the proof that the program computes the inner product of perturbed vectors $\bar{x}[1:n]$ and $\bar{y}[1:n]$, we need only apply conventional error analysis to the mathematical object described by $P \ \& \ k = n$. In fact, once we have verified that the program will terminate and establish $P \ \& \ k = n$, the program plays no further role in the analysis: all subsequent analysis is performed upon the static, mathematical object described by $P \ \& \ k = n$.

Backward Error Analysis:

$$P \Rightarrow [\ 0 \le k \le n \ \& \ z = z^{(k)} \ \&$$

for some sequence $\{\delta_3^{(j)}\}_{j=1}^{k}$

such that $(1-\lambda)^{k-j+2} \le 1+\delta_3^{(j)} \le (1+\mu)^{k-j+2}$:

$$z^{(k)} = \begin{cases} 0 & \text{if } k = 0 \\ \sum_{j=1}^{k} x[j]\, y[j]\, (1+\delta_3^{(j)}) & \text{if } k \ne 0 \end{cases} \quad]$$

If we define

$$\bar{x}[j] \equiv x[j]\,(1+\delta_4^{(j)}) \equiv x[j]\,(1+\delta_3^{(j)})^{1/2} \qquad \text{for } 1 \le j \le n$$

$$\bar{y}[j] \equiv y[j]\,(1+\delta_4^{(j)}) \equiv y[j]\,(1+\delta_3^{(j)})^{1/2} \qquad \text{for } 1 \le j \le n$$

then $z^{(k)}$ may be rewritten as

$$z^{(k)} = \begin{cases} 0 & \text{if } k = 0 \\ \sum_{j=1}^{k} \bar{x}[j]\, \bar{y}[j] & \text{if } 1 \le k \le n \end{cases}$$

Note that $P \ \& \ k = n \Rightarrow z = z^{(n)}$. Since we have shown that the given program is guaranteed to terminate with $P \ \& \ k = n$ valid, we now see that this result may be considered as the exact inner product of the perturbed vectors $\bar{x}[1:n]$ and $\bar{y}[1:n]$. Bounds on the error in $\bar{x}$ and $\bar{y}$ are given above.

QED

Example: Given fp x and y, determine conditions sufficient to guarantee $w = z$ upon termination of the following program.

Program:

$$t := x;$$
$$w := t \oplus y;$$
$$z := x \oplus y$$

If neither overflow nor underflow occurs, we would expect in all cases that $w = z$ upon termination. However, let us see what happens if we just apply axiom A3' repetitively:

$$\begin{aligned} P &\equiv wp(\text{"}z := x \oplus y\text{"},\ w = z) \\ &= (\forall v_1 \in \overline{x \oplus y} : \ w = v_1) \end{aligned}$$

$$\begin{aligned} Q &\equiv wp(\text{"}w := t \oplus y\text{"},\ P) \\ &= (\forall v_2 \in \overline{t \oplus y} : \ (\forall v_1 \in \overline{x \oplus y} : \ v_2 = v_1)) \end{aligned}$$

$$\begin{aligned} R &\equiv wp(\text{"}t := x\text{"},\ Q) \\ &= (\forall v_2 \in \overline{x \oplus y} : \ (\forall v_1 \in \overline{x \oplus y} : \ v_2 = v_1)) \end{aligned}$$

We see that R is valid if and only if $\overline{x \oplus y}$ is a singleton set or empty. By definition of $\overline{x \oplus y}$, this occurs if and only if the interval with endpoints $(x+y)(1-\lambda)$, $(x+y)(1+\mu)$ contains no more than one fp number. (For example, it occurs when $\lambda = \mu = 0$.) Also, if "$w := t \oplus y$" is replaced by "$w := y \oplus x$", then a similar treatment shows that the commutativity of $\oplus$ cannot be concluded. Therefore, since we are unable to infer from axiom A3' many of the important cases in which two fp variables are equal, we conclude that this axiom is, by itself, inadequate to capture all the significant properties of fp arithmetic.

To meet the inadequacy of axiom A3', we considered supplying the following as an additional axiom for dealing with fp sets $\bar{E}$.

For any predicate P with free variables v_1 and v_2, and for any fp expression E,

$$(\forall v_1 \in \bar{E} : (\forall v_2 \in \bar{E} : P)) \iff (\forall v_1 \in \bar{E} : P_{v_1}^{v_2})$$

That is, if P holds for all values v_1 and v_2 drawn from the same set, would we be justified in concluding that P is valid with all occurrences of v_2 replaced by v_1, and conversely?

The indicated equivalence is trivial if $\bar{E}$ is empty. For nonempty $\bar{E}$,

$$\begin{aligned} &(\forall v_1 \in \bar{E} : (\forall v_2 \in \bar{E} : P)) \\ &\quad \Rightarrow (\forall v_1 \in \bar{E} : (\exists v_2 = v_1 \in \bar{E} : P_{v_1}^{v_2})) \end{aligned}$$

from which, the implication, "$\Rightarrow$", is established. For "$\Leftarrow$" we give a counterexample. Suppose $P \equiv (v_1 = v_2)$. Applying the proposed axiom, we obtain

$$(\forall v_1 \in \bar{E} : (\forall v_2 \in \bar{E} : v_1 = v_2)) \iff (\forall v_1 \in \bar{E} : v_1 = v_1)$$

Since the right-hand side is a tautology, "$\Rightarrow$" is trivial. However, since the left-hand side is valid only if $\bar{E}$ is a singleton or empty, "$\Leftarrow$" is invalid.

Therefore, we have shown that if the left-hand side of the proposed axiom is valid, we are justified in concluding the right-hand side. However, since the right-hand side is not sufficient to establish the left-hand side, the former may <u>not</u> be considered as a sufficient pre-condition for the latter.

There is something fundamentally inadequate about axiom A3'. It is true that axiom A3' is able to capture the fact that the result of an fp operation is an fp number chosen from an interval, and that the identity of the number is known with no more precision than knowledge of the endpoints of the interval will allow. However, it is incapable of demonstrating that an fp operation delivers identical results each time it is applied to identical operands. That is, axiom A3' is unable to demonstrate that each fp operation is a <u>function</u>.

Consequently, we discard axiom A3', reject the axiom proposed above, and return to our original set of axioms. In the next section, we show how these axioms may be augmented to support a model of fp arithmetic, with the single-valuedness of the fp operations preserved.

3.7. A Second Approach for Proving Program Correctness

In section 3.7, we use the axioms introduced in section 3.5, augmented by axioms for characterizing what appears to be the significant features of fp arithmetic. We present the additional axioms in section 3.7.1 and use them to prove fp programs correct in section 3.7.2. Several theorems about the arithmetic are proved in the Appendix.

3.7.1 Axioms for Floating-Point Arithmetic

> Entia non sunt multiplicanda praeter necessitatem.
> (Entities must not be multiplied beyond what is
> absolutely necessary.) -- Occam's Razor
>
> Everything should be made as simple as possible, but
> not simpler. -- Albert Einstein

Let $\mathbb{R}$ denote the set of real numbers. Let $\mathbb{D}\mathrm{om}$ be a unary predicate defined on fp expressions, such that $\mathbb{D}\mathrm{om}(E)$ is true only for states in which E is well-defined. We assume that the relations, $<$, $\leq$, $=$, $\neq$, $\geq$, and $>$, implemented for fp numbers, are identical to the relations defined on $\mathbb{R}$. For notational convenience, index i is assumed to range over the set $\{1, 2\}$.

We present axioms characterizing: $\mathbb{F}$, CR_i, $\underline{MIN}$, MIN, MAX, $\overline{MAX}$, λ_i, μ_i, $\oplus$, $\ominus$, $\odot$, and $\oslash$. Informally, $\mathbb{F}$ is the set of fp numbers; MAX and MIN are the largest and smallest positive fp numbers; CR_i denotes two cropping functions, each of which maps reals to fp numbers; $\underline{MIN}$ and $\overline{MAX}$ are the underflow and overflow threshholds; and λ_i and μ_i denote the negative and positive relative error bounds for CR_i.

F1. $\mathbb{F}$ is a finite subset of $\mathbb{R}$

F2. $0, 1 \in \mathbb{F}$

F3. $MAX \in \mathbb{F}$ & $(\forall x \in \mathbb{F} : x \leq MAX)$

F4. $MIN \in \mathbb{F}$ & $(\forall x \in \mathbb{F} : x > 0 \Rightarrow x \geq MIN)$

F5. $(\forall x \in \mathbb{R} : CR_i(x) \in \mathbb{F}$ &
$\quad (\forall y, z \in \mathbb{F} : y \leq x \leq z \Rightarrow y \leq CR_i(x) \leq z))$

F6. $0 < \underline{MIN} \leq MIN$ & $MAX \leq \overline{MAX}$

F7. $0 \leq \lambda_i, \mu_i < 1$

F8. $(\forall x \in \mathbb{R} : [\underline{MIN} \leq x \leq \overline{MAX} \Rightarrow x(1-\lambda_i) \leq CR_i(x) \leq x(1+\mu_i)])$

F9. $(\forall x, y \in \mathbb{R} : x \leq y \Rightarrow CR_i(x) \leq CR_i(y))$

F10. $(\forall x \in \mathbb{R} : \ CR_i(-x) = -CR_i(x))$

F11. $(\forall x, y \in \mathbb{F} : \ xy \geq 0 \ \& \ \mathbb{D}\mathrm{om}(x \oplus y) \ \Rightarrow \ x \oplus y = CR_1(x+y))$

F12. $(\forall x, y \in \mathbb{F} : \ xy \leq 0 \ \& \ \mathbb{D}\mathrm{om}(x \oplus y) \ \Rightarrow \ x \oplus y = CR_2(x+y))$

F13. $(\forall x, y \in \mathbb{F} : \ \mathbb{D}\mathrm{om}(x \otimes y) \ \Rightarrow \ x \otimes y = CR_1(xy))$

F14. $(\forall x, y \in \mathbb{F} : \ \mathbb{D}\mathrm{om}(x \oslash y) \ \Rightarrow \ x \oslash y = CR_1(x/y))$

F15. $(\forall x, y \in \mathbb{F} : \ x \ominus y = x \oplus (-y))$

F5 says that the CR_i are <u>faithful</u>, in the sense defined by Dekker [11,12]; F9 and F10 say that the CR_i are monotone and anti-symmetric about 0; F15 defines $\ominus$ in terms of $\oplus$; and F11 - F14 describe how $x \oplus y$, $x \otimes y$, and $x \oslash y$ are functionally related to the exact sum, product, and quotient.

Many analyses (for example, those of Dahlquist and Björck [10], Forsythe, Malcolm and Moler [19], and Sanderson [60]) define all the fp operations in terms of a single cropping function, usually denoted by "fl". However, this idealization is <u>not</u> valid for many implementations of $\oplus$. (For example, it is not valid for either single or double precision arithmetic on the IBM 360 or 370, where the number of guard digits used, g, is equal to 1. Also, it is not valid for chopped arithmetic on the CDC 6000/Cyber-70 series computers, where $g = t$ (single precision) or $g = 0$ (double precision). We conclude section 3.7.1 by giving counterexamples for each of the addition schemes introduced in section 2.5.) If less than $t+1$ guard digits are used, cancellation of leading digits, which is possible for $x \oplus y$ when the signs of x and y differ, may cause the exact sum $x+y$ to be cropped to a different fp number than that obtained from $w \oplus z$, where

$w+z = x+y$, and w, z have the same sign. Thus, these implementations require _two_ cropping functions: one for $x \circledast y$ when the signs of x and y differ, and the second to be used in all other cases.

Virkkunen [67] noted that the exact sum, $x+y$, rounded to a specified number of digits, will not be realized by scheme S_2 with base 2 or base 10 and certain numbers of guard digits. For scheme S_1 he noted that when x and y have opposite signs, the exact sum, chopped to a certain number of digits, will not be realized for any base and any finite number of guard digits, since the entire x is discarded if $|x|$ is sufficiently small. However, we are not requiring that our CR_i yield the same result as chopping or rounding the exact sum, but only that the CR_i are single-valued functions and satisfy axioms F1 - F15. It is believed that this thesis is the first attempt to model all the fp operations in terms of exactly two cropping functions.

We saw in section 2.5 that 100 percent errors are possible from addition schemes S_1 - S_3 when no guard digits are employed. (Recall also, that we have shown 100 percent errors are possible from rounded additions on the CDC 6000/Cyber-70 series computers.) Since errors of this size are not allowed between $\underline{MIN}$ and $\overline{MAX}$ by axioms F7 and F8, at least one guard digit is _required_ of the fp implementations supported by these axioms. Virkkunen [67] has recently shown that one guard digit is _sufficient_ for chopped scheme S_1 and rounded scheme S_2 to satisfy the faithfulness property embodied by axiom F5. Therefore, we conclude that the axioms given above are sufficiently general to hold for either rounded or chopped arithmetic, under a variety of implementations, provided that at least one guard digit is employed for all the fp operations.

Note that the axioms support various treatments of overflow and underflow. If **Dom** is defined to be true everywhere, then the fp operations are closed on **F**: all real numbers $x \geq MAX$ are mapped by CR_i to MAX, and all real numbers x such that $0 \leq x \leq MIN$ are mapped by CR_i to either 0 or MIN. (We prove this in the Appendix.) Alternatively, by defining **Dom** to be false for real $x > \overline{MAX}$, and false for real x such that $0 < x < \underline{MIN}$, overflowed and underflowed operations produce <u>undefined</u> results. Finally, note that by defining $\mathbf{Dom}(x \oslash 0) =$ false, division by zero can be handled in a clean and attractive way.

Knuth gave about 20 desirable properties of fp arithmetic in section 4.2.2 of [42] and in the associated exercises. However, it is apparent that he intended only to demonstrate the desirability of these properties, rather than to present a minimal set of axioms, since from his presentation it is not clear which properties are to be assumed as axioms and which are to be derived as theorems. We show in the appendix that all but two of these properties can be derived from F1 - F15. The two exceptional cases, which are noted in the appendix, arise from the fact that our axioms allow underflowed results to be set to zero, whereas his axioms do not apply if underflow occurs.

Wirth [73] clearly defines 9 axioms for fp arithmetic. However, some of these axioms have subcases. (For example, $x \oplus y = y \oplus x$ and $x \otimes y = y \otimes x$ are presented as one axiom.) By treating each subcase as a separate axiom, one can count 18 distinct axioms. Wirth's axioms allow underflowed results to be set to zero, and with one exception, can all be derived from F1 - F15. His axiom A7,

$$x \geq y \geq 0 \Rightarrow (x \ominus y) \oplus y = x$$

cannot be derived from F1 - F15. In fact, the following counterexample shows that this property does not hold for any of the schemes S_1 - S_3, introduced in section 2.5.

Scheme S_1, with $g \geq 1$ and $\beta \geq 2$:

$$x = \beta^{-1} + \beta^{-t} \qquad y = \frac{1}{2}\beta^{-t}$$

$$(x \ominus y) \oplus y = \beta^{-1} \neq x$$

Schemes S_2 and S_3, with $g \geq 1$ and $\beta \geq 2$:

$$x = \beta^{-1} + \beta^{-t} \qquad y = \frac{1}{2}\beta^{-t}$$

$$(x \ominus y) \oplus y = \beta^{-1} + 2\beta^{-t} \neq x$$

On the grounds of succinctness, we prefer axioms F1 - F15 to the axioms of Knuth and Wirth. But axioms F1 - F15 are also desirable, because they provide a characterization for the processes that give rise to the properties Knuth and Wirth have enumerated.

Brown [6] gives only 4 axioms, and Dekker [12] gives 8 sets of ≤ 4 axioms for characterizing fp arithmetic. (Dekker's axiom sets differ on their assumptions concerning underflow, overflow, precision, faithfulness, and whether rounding or chopping is used.) Even though the axiom sets proposed by Brown and Dekker are quite succinct, they are unable to demonstrate that fp addition and multiplication are commutative, and that each of the fp operators is monotone and anti-symmetric about zero. In fact, we find these axioms to be inadequate for the same reason that we discarded axiom A3' in section 3.6. That is, they are unable to demonstrate the fact that regardless of whether the arithmetic rounds or chops, each fp operation delivers identical results each time it is applied to identical operands. Since we consider this property to be significant from the standpoint of proving programs correct, we prefer F1 - F15 to any of the axiom sets proposed by Brown or Dekker.

In summary, we believe that F1 - F15 provide a characterization of fp arithmetic at the proper level of abstraction for reasoning about implementation independent properties of fp programs.

For each of the addition schemes introduced in section 2, we now give counterexamples to indicate where $\oplus$ cannot be modeled by a single function of the exact sum.

Scheme S_1

with $0 \le g \le t$ and $\beta \ge 2$:

$x = \beta^{-1}$ $\qquad y = -\beta^{-g-t-1}$

$w = (1 - \beta^{-t})\beta^{-1}$ $\qquad z = (1 - \beta^{-g})\beta^{-t-1}$

$x + y = w + z = \beta^{-1} - \beta^{-g-t-1}$

but, $x \oplus y = \beta^{-1}$ and $w \oplus z = \beta^{-1} - \beta^{-t-1}$

Scheme S_2

with $g = 0$ and $\beta > 2$:

(Same as for scheme S_1 with $g = 0$ and $\beta > 2$.)

with $g = 0$ and $\beta = 2$:

$x = 2^{-1}$ $\quad$ $y = -\,2^{-t-1}$

$w = (1 - 2^{-t})\,2^{-1}$ $\quad$ $z = 0$

$x + y = w + z = 2^{-1} - 2^{-t-1}$

but, $x \oplus y = 2^{-1} - 2^{-t}$ and $w \oplus z = 2^{-1} - 2^{-t-1}$

with $g = 1$ and $\beta \geq 2$:

$x = \beta^{-1}$ $\quad$ $y = -\frac{1}{2}\beta^{-t-1}$

$w = (1 - \beta^{-t})\,\beta^{-1}$ $\quad$ $z = \frac{1}{2}\beta^{-t-1}$

$x + y = w + z = \beta^{-1} - \frac{1}{2}\beta^{-t-1}$

but, $x \oplus y = \beta^{-1} - \beta^{-t-1}$ and $w \oplus z = \beta^{-1}$

with $2 \leq g \leq t$ and $\beta > 2$:

$x = \beta^{-1}$ $\quad$ $y = -(\frac{1}{2} + \beta^{-g})\,\beta^{-t-1}$

$w = (1 - \beta^{-t})\,\beta^{-1}$ $\quad$ $z = (\frac{1}{2} - \beta^{-g})\,\beta^{-t-1}$

$x + y = w + z = \beta^{-1} - \frac{1}{2}\beta^{-t-1} - \beta^{-t-g-1}$

but, $x \oplus y = \beta^{-1}$ and $w \oplus z = \beta^{-1} - \beta^{-t-1}$

with $2 \leq g \leq t - 1$ and $\beta = 2$:

$x = 2^{-1} + 2^{-t}$ $\quad$ $y = -(1 + 2^{-g})\,2^{-t-1}$

$w = 2^{-1}$ $\quad$ $z = (1 - 2^{-g})\,2^{-t-1}$

$x + y = w + z = 2^{-1} + 2^{-t-1} - 2^{-g-t-1}$

but, $x \oplus y = 2^{-1}$ and $w \oplus z = 2^{-1} + 2^{-t}$

Scheme S_3

with $g = 0$ and $\beta \geq 2$:

(Same as for scheme S_1 with $g = 0$ and $\beta \geq 2$.)

with $g = 1$ and $\beta > 2$:

$x = \beta^{-1}$ $\qquad y = -(\beta - 1)\beta^{-t-2}$

$w = (1 - \beta^{-t})\beta^{-1}$ $\qquad z = \beta^{-t-2}$

$x + y = w + z = \beta^{-1} - \beta^{-t-1} + \beta^{-t-2}$

but, $x \oplus y = \beta^{-1}$ and $w \oplus z = \beta^{-1} - \beta^{-t-1}$

with $g = 1$ and $\beta = 2$:

$x = 2^{-1}$ $\qquad y = -(1 - 2^{-3})2^{-t}$

$w = 2^{-1} - 2^{-t}$ $\qquad z = 2^{-t-3}$

$x + y = w + z = 2^{-1} - 2^{-t} + 2^{-t-3}$

but, $x \oplus y = 2^{-1} - 2^{-t-1}$ and $w \oplus z = 2^{-1} - 2^{-t}$

with $2 \leq g \leq t$ and $\beta \geq 2$:

(Same as for scheme S_2 with $2 \leq g \leq t$ and $\beta > 2$.)

Suppose that $\oplus$ is performed by any of the schemes S_1, S_2, or S_3, with $g > t$ guard digits. We show that in these cases, $\oplus$ <u>can</u> be modeled by a single function of the exact sum.

Note that the exact sum of fp x and y, chopped or rounded to a specified number of digits, will be formed by $x \oplus y$ as long as no digits are lost during pre-adjustment. That is, the chopped or rounded t-digit sum $x \oplus y$ is certain to be a function of the exact sum $x + y$, as long as the exponents of x and y differ by an amount $\leq g$.

For $g > t$, it is sufficient to show that $\oplus$ can be modeled by a single function of the exact sum of fp x and y, where the exponents differ by an amount $\geq t + 2$, and one or more nonzero digits is lost during pre-adjustment. Suppose that the signs of x and y differ. Under the stated conditions, we find that after addition, but prior to post-adjustment shifting:

(1) the leftmost first or second digit in the sum is nonzero;

(2) digit positions $t + 1$ and $t + 2$ contain digits of size $\beta - 1$; and,

(3) some nonzero digit to the right of position $t + g$ in the exact sum was lost during pre-adjustment.

Given conditions (1) - (3), we see that the exact sum $x + y$ has $\geq 2t + 1$ significant digits, with a nonzero digit t places to the right of the most significant digit position. Since there are no fp w and z with the same sign whose exact sum can satisfy these conditions, the desired result follows. Once again, note that this result is independent of base and cropping scheme.

It is also true that $\oplus$ can be modeled by a single function of the exact sum, in the peculiar case of scheme S_2, where $\beta = 2$ and $g = t$. As noted above, $x \oplus y$ is certain to be a function of $x + y$ as long as no digits are lost during pre-adjustment. Suppose that fp x and y have opposite signs, that their exponents differ by an amount $\geq t + 1$, and that some nonzero digit is shifted out of the guard digit registers during pre-adjustment. If the exact sum $x + y$ contains $\geq 2t + 1$ significant digits, then an argument similar to that given above shows that there are no fp w and z with the same sign, such that $w + z = x + y$. Otherwise, $x + y$ will contain exactly 2t significant digits. Under the stated conditions, it can be shown that the pre-adjustment round causes the addition step of $x \oplus y$ to deliver the same leading $t + 2$ digits as occur in the exact sum $x + y$. Consequently, for any fp w and z with the same sign, such that $w + z = x + y$, forming $w \oplus z$ incurs no pre-adjustment rounding, and the addition step here also, produces the same $t + 2$ leading digits as the exact sum $x + y$. The desired result follows.

3.7.2 Correctness Proofs

In this section we show how axioms A1 - A5, together with axioms F1 - F15, can be used to reason about the correctness of fp programs. Theorems refered to in this section with the prefix "TH" are proved from F1 - F15 in the appendix.

Consider the program that was inadequately handled by our first approach for proving program correctness.

__Example__: Given fp x and y, determine conditions sufficient to guarantee $w = z$ upon termination of the following program.

<u>Program</u>:

```
t := x;
w := t ⊕ y;
z := x ⊕ y
```

Applying axiom A3 repetitively, we obtain:

$$
\begin{aligned}
P &\equiv wp(\text{"}z := x \oplus y\text{"},\ w = z) \\
&= \mathbf{Dom}(x \oplus y)\ \&\ w = x \oplus y \\[1em]
Q &\equiv wp(\text{"}w := t \oplus y\text{"},\ P) \\
&= \mathbf{Dom}(t \oplus y)\ \&\ \mathbf{Dom}(x \oplus y)\ \&\ t \oplus y = x \oplus y \\[1em]
R &\equiv wp(\text{"}t := x\text{"},\ Q) \\
&= \mathbf{Dom}(x)\ \&\ \mathbf{Dom}(x \oplus y)\ \&\ \mathbf{Dom}(x \oplus y)\ \&\ x \oplus y = x \oplus y \\
&= \mathbf{Dom}(x \oplus y)
\end{aligned}
$$

That is, the given program will terminate with $w = z$, provided only that x and y are initialized so that $x \oplus y$ is defined. This is the result we desired. <u>QED</u>

Consider the inner product program introduced in section 3.6. Compare the following treatment with that of section 3.6. Also, note how the **Dom** predicate lends itself to reasoning about overflow, underflow, and bounds on the relative error.

<u>Example</u>: Given $n \geq 0$ and fp vectors x[1:n] and y[1:n]. Use fp arithmetic to determine the <u>exact</u> inner product of $\bar{x}$[1:n] and $\bar{y}$[1:n], where $\bar{x}$ and $\bar{y}$ are perturbed versions of x and y. Bound the amount by which $\bar{x}$ and $\bar{y}$ deviate from x and y.

<u>Program</u>:

$$
\begin{array}{l}
z, k := 0, 0; \ \{ P \} \\
\textbf{do} \ \ k \neq n \ \rightarrow \ \{ P \ \ \& \ \ k \neq n \} \\
\qquad\qquad\qquad k := k + 1; \\
\qquad\qquad\qquad z := z \oplus x[k] \otimes y[k] \quad \{ P \} \\
\textbf{od}
\end{array}
$$

In order to direct our attention to fp arithmetic, we define

$$(\forall j : \mathbb{D}\text{om}(j+1) \equiv \text{true})$$

Also, we define

$$z^{(j)} \equiv \begin{cases} 0 & \text{if} \quad j = 0 \\ z^{(j-1)} \oplus x[j] \otimes y[j] & \text{if} \quad 1 \leq j \leq n \end{cases}$$

$$P \quad \equiv [\, 0 \leq k \leq n \ \ \& \ \ z = z^{(k)} \,]$$

Predicate P gives bounds on k and states that variable z is the kth element of some sequence $\{ z^{(j)} \}_{j=0}^{n}$, where $z^{(j)}$ is defined in terms of $z^{(j-1)}$, x[j], y[j], and <u>fp operators</u> $\oplus$, and $\otimes$.

We now derive conditions which are sufficient to ensure that predicate P is <u>invariant</u> with respect to the **do** ··· **od** loop in the given program.

<u>Derivation of Conditions to Ensure the Invariance of P</u>:

It is trivial to see that P holds immediately before executing the **do** ··· **od** loop, and that k ≠ n is valid if the loop guard is found to be true.

If P is to hold at the end of the loop, for each iteration, then by axiom A5, the following must hold at the beginning of the loop, for each iteration.

$$Q \equiv wp(\text{"}k := k + 1\text{"},\ R)$$

where $$R \equiv wp(\text{"}z := z \oplus x[k] \circledast y[k]\text{"},\ P)$$

Since, by definition, $\mathbf{D}om(k + 1) = true$, applying axiom A3 to Q yields

$$Q = [\ \mathbf{D}om(E)\ \&\ P^z_E\ ]^k_{k+1}$$

where $$E \equiv z \oplus x[k] \circledast y[k]$$

Therefore, by performing the indicated substitutions in Q and applying theorem 1, we see that the invariance of P is established if

$$P\ \&\ k \neq n \Rightarrow Q$$

where

$$\begin{aligned} Q = {} & \mathbf{D}om(z \oplus x[k+1] \circledast y[k+1])\ \& \\ & 0 \le k + 1 \le n\ \&\ z \oplus x[k+1] \circledast y[k+1] = z^{(k+1)} \end{aligned}$$

Note that

$$\begin{aligned} P\ \&\ k \neq n & \Rightarrow 0 \le k < n \\ & \Rightarrow 1 \le k + 1 \le n \end{aligned}$$

For $1 \le k + 1 \le n$, the definition of $z^{(k+1)}$ yields

$$z^{(k+1)} = z^{(k)} \oplus x[k+1] \circledast y[k+1]$$

Finally, from the definition of P, we see that $P\ \&\ k \neq n \Rightarrow Q$, provided

$$0 \le k < n \Rightarrow \mathbf{D}om(z^{(k)} \oplus x[k+1] \circledast y[k+1])$$

That is, P is guaranteed to be invariant with respect to the **do** ··· **od** loop, provided the above condition holds among the x and y vectors.

If the above condition holds for each k, then since k is incremented using exact arithmetic, the program will terminate attaining the assertion

$$P \ \& \ k = n$$

QED

Backward Error Analysis:

Let us define

$$(\forall v, w \in \mathbb{F} : \ \mathbf{Dom}(v \oplus w) \ \equiv \ [\ \underline{MIN} \le v + w \le \overline{MAX}\ ]\ )$$

$$(\forall v, w \in \mathbb{F} : \ \mathbf{Dom}(v \otimes w) \ \equiv \ [\ \underline{MIN} \le \ vw \ \le \overline{MAX}\ ]\ )$$

$$\bar{\lambda} \ \equiv \ \max\{\lambda_i\}, \qquad \bar{\mu} \ \equiv \ \max\{\mu_i\} \qquad \text{for} \quad i = 1, 2$$

With these definitions, axioms F8 and F11 - F13 give us

$$1 \le j \le n \ \& \ \mathbf{Dom}(z^{(j-1)} \oplus x[j] \otimes y[j]) \ \Rightarrow$$

$$\exists\, \delta_1^{(j)}, \delta_2^{(j)} \ : \ -\bar{\lambda} \le \delta_1^{(j)}, \delta_2^{(j)} \le \bar{\mu} \ :$$

$$z^{(j)} = (\, z^{(j-1)} + x[j]\, y[j]\, (1 + \delta_1^{(j)}))\, (1 + \delta_2^{(j)})$$

Consequently, by induction on j, we obtain

$$P \ \& \ [\,\forall j : \ 0 \le j < n : \tag{3.7.2.1}$$

$$\mathbf{Dom}(\, z^{(j)} \oplus x[j+1] \otimes y[j+1]\,) \ ]$$

$$\Rightarrow \ [\ 0 \le k \le n \ \& \ z = z^{(k)} \ \&$$

$$\text{for some sequence } \{\delta_3^{(j)}\}_{j=1}^{k}$$

$$\text{such that } (1 - \bar{\lambda})^{k-j+2} \le 1 + \delta_3^{(j)} \le (1 + \bar{\mu})^{k-j+2} \ :$$

$$z^{(k)} = \begin{cases} 0 & \text{if } k = 0 \\ \sum_{j=1}^{k} x[j]\, y[j]\, (1 + \delta_3^{(j)}) & \text{if } 1 \le k \le n \end{cases} \quad]$$

From this point, the error analysis is identical to that given in section 3.6. <u>QED</u>

From (3.7.2.1) and the definition of **Dom**, given above, it is straightforward to determine conditions sufficient to preclude overflow and underflow. For example, if

$$(\forall j : 1 \le j \le n : x[j]\, y[j] \ge 0)$$

then neither overflow nor underflow will occur, provided

$$\sum_{j=1}^{n} x[j]\, y[j]\, (1 + \overline{\mu})^{n-j+2} \le \overline{\text{MAX}} \quad \&$$

$$\sum_{j=1}^{n} x[j]\, y[j]\, (1 - \overline{\lambda})^{n-j+2} \ge \underline{\text{MIN}}$$

We leave further analysis of the static, mathematical object described by P & k = n to the numerical analysts.

In the following example, note how the inexactness of fp arithmetic is taken into account in reasoning about termination.

<u>Example</u>: Suppose f is continuous in $[x_0, y_0]$, $0 < x_0 < y_0$, and $f(x_0) < 0 < f(y_0)$. Determine an x and a y such that for some α and some $\eta > 0$,

$$x \le \alpha \le y \quad \& \quad f(\alpha) = 0 \quad \& \quad y - x \le \eta$$

<u>Program</u>:

$$
\begin{array}{l}
x, y := x_0, y_0; \\
\{P\} \\
\textbf{do}\ \ y \ominus x > \varepsilon \rightarrow \\
\quad \{P \ \&\ y \ominus x > \varepsilon\} \\
\quad z := E; \\
\quad \textbf{if}\ \ f(z) \le 0 \rightarrow x := z \\
\quad \square\ \ f(z) \ge 0 \rightarrow y := z \\
\quad \textbf{fi} \\
\quad \{P\} \\
\textbf{od}
\end{array}
$$

where E is one of

$$
\begin{array}{lcl}
E_1 & \equiv & x \oplus (y \ominus x) \oslash 2 \\
E_2 & \equiv & x \ominus (x \ominus y) \oslash 2 \\
E_3 & \equiv & y \ominus (y \ominus x) \oslash 2 \\
E_4 & \equiv & y \oplus (x \ominus y) \oslash 2 \\
E_5 & \equiv & (x \oslash 2) \oplus (y \oslash 2) \\
E_6 & \equiv & (x \oplus y) \oslash 2
\end{array}
$$

and, for the present, we assume that f is computed exactly.

In order to direct our attention to the effect of fp arithmetic upon termination (or convergence) of the preceding program, we define

$$(\forall j : 1 \le j \le 6 : \mathrm{Dom}(E_j) \equiv \text{true})$$

Also, we define

$$P \equiv [\, 0 < x_0 \le x \le y \le y_0 \ \ \&\ \ f(x) \le 0 \le f(y) \,]$$

For each choice of E, we now derive conditions sufficient to ensure that predicate P is invariant with respect to the **do** ⋯ **od** loop in the given program. That is, these conditions will guarantee P to be valid immediately before the first iteration and after each subsequent iteration. Clearly, the second conjunct in P guarantees a root of f in the interval [x, y].

In the previous example, the use of sequences, defined in terms of fp operations, freed the proof of the invariance of P from any consideration of error in the fp operations. In fact, with this approach, the invariance proof demonstrates only that the fp operations are applied under the conditions and in the order dictated by statically specified sequences. However, we saw that such an approach requires an error analysis to be performed inductively upon these sequences, when the inexactness of the fp operations must finally be taken into account.

In the present example, P is not defined in terms of sequences, but only in terms of the initial and current values of x and y. With this approach, the invariance proof is more complicated, because it must take into account the inexactness of fp arithmetic. However, a separate induction during the error analysis is not needed.

Derivation of Conditions to Ensure the Invariance of P:

It is trivial to see that P holds immediately before executing the **do** ⋯ **od** loop, and that $y \ominus x > \varepsilon$ is valid if the loop guard is found to be true.

If P is to hold at the end of the loop, for each iteration, then by axiom A5, the following must hold at the beginning of the loop, for each iteration.

$$R \equiv wp(\text{"}z := E\text{"},\ Q)$$

where, by axiom A4,

$$Q \equiv [\ f(z) \le 0 \Rightarrow wp(\text{"}x := z\text{"},\ P)\ \&\ f(z) \ge 0 \Rightarrow wp(\text{"}y := z\text{"},\ P)\ ]$$

and, by axiom A3,

$$wp(\text{"}x := z\text{"},\ P) = P^x_z$$
$$wp(\text{"}y := z\text{"},\ P) = P^y_z$$
$$wp(\text{"}z := E\text{"},\ Q) = Q^z_E$$

Therefore, by theorem 1, the invariance of P is established if

$$P\ \&\ y \ominus x > \varepsilon \Rightarrow R$$

where

$$\begin{aligned} R = [\ & f(E) \le 0 \Rightarrow \\ & 0 < x_0 \le E \le y \le y_0\ \&\ f(E) \le 0 \le f(y)\ ]\ \& \\ [\ & f(E) \ge 0 \Rightarrow \\ & 0 < x_0 \le x \le E \le y_0\ \&\ f(x) \le 0 \le f(E)\ ] \end{aligned}$$

Note that

$$P\ \&\ x \le E \le y \Rightarrow R$$

Consequently, the invariance of P will be established for each choice of E if we can show

$$P\ \&\ y \ominus x > \varepsilon \Rightarrow x \le E \le y \qquad (3.7.2.2)$$

We shall see that (3.7.2.2) follows from our proof that the given program terminates.

From theorem 1, we see that program termination is guaranteed provided we can determine a t, defined on a discrete set, such that

$$P \;\&\; y \ominus x > \varepsilon \;\Rightarrow\; t \geq 0 \qquad (3.7.2.3)$$

and

$$P \;\&\; y \ominus x > \varepsilon \;\Rightarrow\; wp(\text{"}T := t;\ SL\text{"},\ t < T) \qquad (3.7.2.4)$$

where "SL" denotes the body of the **do** ··· **od** loop, and T is some variable not appearing in SL.

Define

$$t \equiv y - x$$

Note that t is the exact difference between program variables x and y. Since $x, y \in \mathbb{F}$, it follows that x, y, and t take on values from discrete sets. With this t, conditions (3.7.2.3) and (3.7.2.4) imply that the number of fp values between x and y strictly decrease each iteration, with x remaining $\leq$ y. Clearly, these conditions are sufficient to guarantee termination.

We now derive conditions sufficient to establish (3.7.2.3) and (3.7.2.4).

<u>Derivation of Conditions to Ensure Termination</u>:

Note that $P \Rightarrow x \leq y \Rightarrow t \geq 0$, which establishes (3.7.2.3).

The derivation of wp(SL, t < T) is identical to the derivation of wp(SL, P), given above. We find that

$$wp(SL,\ t < T) = \qquad (3.7.2.5)$$

$$[\ f(E) \le 0 \ \Rightarrow\ y - E < T\ ]\ \ \&$$
$$[\ f(E) \ge 0 \ \Rightarrow\ E - x < T\ ]$$

Thus, by axioms A3 and A5,

$$wp(\text{"}T := t;\ SL\text{"},\ t < T) =$$
$$[\ f(E) \le 0 \ \Rightarrow\ y - E < y - x\ ]\ \ \&$$
$$[\ f(E) \ge 0 \ \Rightarrow\ E - x < y - x\ ]$$

In summary, (3.7.2.4) will be established and termination guaranteed if we can show

$$P\ \ \&\ \ y \ominus x > \varepsilon \ \Rightarrow\ x < E < y \qquad (3.7.2.6)$$

Note that the condition we derived to demonstrate the invariance of P, (3.7.2.2), follows a fortiori, from (3.7.2.6).

From axioms F10, F12, F14, and F15 we see that

$$[\ \forall x, y \in \mathbb{F} :\ E_1 = x \oplus (y \ominus x) \oslash 2 = x \ominus (x \ominus y) \oslash 2 = E_2\ ]$$

$$[\ \forall x, y \in \mathbb{F} :\ E_3 = y \ominus (y \ominus x) \oslash 2 = y \oplus (x \ominus y) \oslash 2 = E_4\ ]$$

Consequently, for all possible fp x and y, E_1 delivers _exactly_ the same results and errors as E_2, and E_3 delivers _exactly_ the same results and errors as E_4. In other words, E_1 is _functionally equivalent_ to E_2, and E_3 is _functionally equivalent_ to E_4.

Henceforth, we direct our attention to showing (3.7.2.6) for E_1, E_3, E_5, and E_6.

Lemma 1: If $y - x \ge \max\{MIN,\ 2(1-\lambda_2)^{-1}\underline{MIN}\}$ then the relative error bounds λ_i, μ_i, given by axiom F8 are applicable to E_1.

<u>Proof of Lemma</u>:

Note that the bounds λ_i, μ_i, given by F8 are applicable to

$$E_1 = x \oplus (y \ominus x) \oslash 2$$

provided that

$$(1)\ \underline{MIN} \le y - x \le \overline{MAX},$$

$$(2)\ \underline{MIN} \le \frac{1}{2}(y \ominus x) \le \overline{MAX}, \text{ and}$$

$$(3)\ \underline{MIN} \le x + (y \ominus x) \oslash 2 \le \overline{MAX}$$

By F3, F4, and $P \Rightarrow (0 < x \le y)$, we obtain

$$MIN \le x \le y \le MAX \qquad (3.7.2.7)$$

By hypothesis, $y - x \ge MIN$. Thus, by axiom F6 and (3.7.2.7), (1) is established.

Since (1) is valid, it follows by axioms F8, F12, and F15 that

$$(y - x) (1 - \lambda_2) \le y \ominus x \qquad (3.7.2.8)$$

By hypothesis, $y - x \ge 2 (1 - \lambda_2)^{-1} \underline{MIN}$. Thus, by (3.7.2.8)

$$\underline{MIN} \le \frac{1}{2}(y \ominus x)$$

From (3.7.2.7) we see that $y - x \le MAX$. Thus, by axioms F5, F12, and F15

$$\frac{1}{2}(y \ominus x) \le \overline{MAX}$$

which establishes (2).

We have noted that $MIN \le y - x \le MAX$. Thus, by (3.7.2.1) in the proof of TH40,

$$0 \le (y \ominus x) \oslash 2 \le y - x \tag{3.7.2.9}$$

Finally, (3) follows from (3.7.2.7) and (3.7.2.9). QED

Lemma 2: If $y - x \ge \max\{\mathrm{MIN}, 2(1-\lambda_2)^{-1}\underline{\mathrm{MIN}}\}$ then the relative error bounds λ_i, μ_i, given by axiom F8 are applicable to E_3.

Proof of Lemma:

Similar to the proof of lemma 1. QED

Lemma 3: If $x \ge 2\,\underline{\mathrm{MIN}}$ and $(y-x)/x \ge 2\mu_1/(1-\mu_1)$ then the relative error bounds λ_i, μ_i, given by axiom F8 are applicable to E_5.

Proof of Lemma:

Note that the bounds λ_i, μ_i, given by F8 are applicable to

$$E_5 = (x \oslash 2) \oplus (y \oslash 2)$$

provided that

(1) $\underline{\mathrm{MIN}} \le \frac{1}{2}x,\ \frac{1}{2}y \le \overline{\mathrm{MAX}}$, and

(2) $\underline{\mathrm{MIN}} \le x \oslash 2 + y \oslash 2 \le \overline{\mathrm{MAX}}$

By hypothesis, $x \ge 2\,\underline{\mathrm{MIN}}$. Thus, by axiom F6 and (3.7.2.7), (1) is established.

Since (1) is valid, it follows by axioms F8 and F14 that

$$x \oslash 2 + y \oslash 2 \le \frac{1}{2}(x+y)(1+\mu_1) \tag{3.7.2.10}$$

By hypothesis, $(y-x)/x \ge 2\mu_1/(1-\mu_1)$. Rearranging, we find that this is equivalent to

$$x + y \le 2(1+\mu_1)^{-1}y \tag{3.7.2.11}$$

Thus, by (3.7.2.7), (3.7.2.10), (3.7.2.11), and axiom F6,

$$x \oslash 2 + y \oslash 2 \leq \overline{MAX}$$

Also, since (1) is valid, it follows from TH2.5, F5, F6, and F14 that

$$\underline{MIN} \leq x \oslash 2 + y \oslash 2$$

which establishes (2). QED

Lemma 4: If $x + y \leq \overline{MAX}$ and $(y - x)/x \geq 2\lambda_1 / (1 - \lambda_1)$ then the relative error bounds λ_i, μ_i, given by axiom F8 are applicable to E_6.

Proof of Lemma:

Note that the bounds λ_i, μ_i, given by F8 are applicable to

$$E_6 = (x \oplus y) \oslash 2$$

provided that

(1) $\underline{MIN} \leq x + y \leq \overline{MAX}$, and

(2) $\underline{MIN} \leq \frac{1}{2}(x \oplus y) \leq \overline{MAX}$

By hypothesis, $x + y \leq \overline{MAX}$. Thus, by axiom F6 and (3.7.2.7), (1) is established.

Since (1) is valid, it follows by axioms F8 and F11 that

$$(x + y)(1 - \lambda_1) \leq x \oplus y \tag{3.7.2.12}$$

By hypothesis, $(y - x)/x \geq 2\lambda_1 / (1 - \lambda_1)$. Rearranging, we find that this is equivalent to

$$x + y \geq 2(1 - \lambda_1)^{-1} x \tag{3.7.2.13}$$

Thus, by (3.7.2.7), (3.7.2.12), (3.7.2.13), and axiom F6,

$$\underline{MIN} \le \frac{1}{2}(x \oplus y)$$

Finally, by (3.7.2.7) and axioms F3, F5, F6, and F11,

$$x \oplus y \le \overline{MAX}$$

which establishes (2). QED

Now, given that lemmas 1-4 are satisfied, by (3.7.2.7) and axioms F8, F11, F12, F14, and F15, we obtain

$$[x + \frac{1}{2}(y - x)(1 - \lambda_2)(1 - \lambda_1)](1 - \lambda_1) \le E_1 \le [x + \frac{1}{2}(y - x)(1 + \mu_2)(1 + \mu_1)](1 + \mu_1) \quad (3.7.2.14)$$

$$[y - \frac{1}{2}(y - x)(1 + \mu_2)(1 + \mu_1)](1 - \lambda_2) \le E_3 \le [y - \frac{1}{2}(y - x)(1 - \lambda_2)(1 - \lambda_1)](1 + \mu_2) \quad (3.7.2.15)$$

$$[\frac{1}{2}x(1 - \lambda_1) + \frac{1}{2}y(1 - \lambda_1)](1 - \lambda_1) \le E_5 \le [\frac{1}{2}x(1 + \mu_1) + \frac{1}{2}y(1 + \mu_1)](1 + \mu_1) \quad (3.7.2.16)$$

$$\frac{1}{2}(x + y)(1 - \lambda_1)^2 \le E_6 \le \frac{1}{2}(x + y)(1 + \mu_1)^2 \quad (3.7.2.17)$$

Note that the bounds on E_5, given by (3.7.2.16), are identical to the bounds on E_6, given by (3.7.2.17). Thus, even though E_5 and E_6 are not functionally equivalent, our analysis shows that, for fp x and y satisfying lemmas 3 and 4, E_5 and E_6 are equally precise expressions for computing the midpoint of x and y.

From (3.7.2.14) we see that (3.7.2.5) can be guaranteed for E_1, provided

$$x < [x + \frac{1}{2}(y - x)(1 - \lambda_2)(1 - \lambda_1)](1 - \lambda_1) \quad \& \quad (3.7.2.18)$$

$$y > [x + \tfrac{1}{2}(y - x)(1 + \mu_2)(1 + \mu_1)](1 + \mu_1)$$

After some algebra, (3.7.2.18) yields

$$\frac{y - x}{x} > \frac{2\lambda_1}{(1 - \lambda_2)(1 - \lambda_1)^2} \quad \& \qquad (3.7.2.19)$$

$$(\bar{\mu} < 2^{1/3} - 1) \Rightarrow \frac{y - x}{x} > \frac{2\mu_1}{2 - (1 + \mu_2)(1 + \mu_1)^2}$$

where $\bar{\mu} \equiv \max\{\mu_1, \mu_2\}$. Let $\bar{\lambda} \equiv \max\{\lambda_1, \lambda_2\}$. Throughout the remaining analysis, we assume both $\bar{\mu} < 2^{1/3} - 1$ and $\bar{\lambda} < 2^{1/3} - 1$. These assumptions are reasonable for either rounded or chopped arithmetic in any base that uses ≥ 3 digit numbers and at least one guard digit.

It is interesting to note that if chopped arithmetic is used, then $\mu_1 = 0$, and the second conjunct of (3.7.2.19) becomes $y > x$. Thus, if chopped arithmetic is used, and the conditions of lemma 1 are satisfied, then E_1 is <u>guaranteed</u> to deliver a value strictly less than y: positive relative errors, including those that may result from forming $y \ominus x$, are of no consequence for ensuring $E_1 < y$.

For E_3, a derivation similar to that which gave (3.7.2.19) yields in this case

$$(\bar{\mu} < 2^{1/2} - 1) \Rightarrow \frac{y - x}{x} > \frac{2\lambda_2}{(1 - \lambda_2)[2 - (1 + \mu_2)(1 + \mu_1)]} \quad \& \qquad (3.7.2.20)$$

$$(\bar{\lambda} < 1 - (2\mu_2)^{1/2}) \Rightarrow \frac{y - x}{x} > \frac{2\mu_2}{(1 - \lambda_2)(1 - \lambda_1)(1 + \mu_2) - 2\mu_2}$$

Note that since we are assuming both $\bar{\mu} < 2^{1/3} - 1$ and $\bar{\lambda} < 2^{1/3} - 1$, the consequents of the preceding implications apply.

If chopped arithmetic is used to form E_3, then as before, we observe $\mu_1 = 0$. However, in contrast with E_1, we cannot conclude any conditions sufficient to guarantee $E_3 < y$ that are independent of positive relative errors.

For both E_5 and E_6, a derivation similar to that which gave (3.7.2.19), yields in this case

$$\frac{y-x}{x} > \frac{2\lambda_1(2-\lambda_1)}{(1-\lambda_1)^2} \quad \& \tag{3.7.2.21}$$

$$(\mu_1 < 2^{1/2} - 1) \Rightarrow \frac{y-x}{x} > \frac{2\mu_1(2+\mu_1)}{2-(1+\mu_1)^2}$$

Since we are assuming $\bar{\mu} < 2^{1/3} - 1$, the consequent of the latter conjunct follows.

It is interesting to note that the conditions

$$\frac{y-x}{x} \geq \frac{2\lambda_1}{1-\lambda_1} \quad \& \quad \frac{y-x}{x} \geq \frac{2\mu_1}{1-\mu_1}$$

from lemmas 3 and 4 follow, a fortiori, from (3.7.2.21).

If chopped arithmetic is used to form E_5 and E_6, then $\mu_1 = 0$, and the second conjunct of (3.7.2.21) becomes $y > x$, as we saw for E_1. Thus, E_5 is <u>guaranteed</u> to deliver a value strictly less than y, provided only $y > x \geq 2\,\underline{\text{MIN}}$; E_6 is <u>guaranteed</u> to deliver a value strictly less than y provided only $y > x$ & $x + y \leq \underline{\text{MAX}}$.

Summarizing, we obtain the following conditions which are sufficient to establish (3.7.2.5).

For E_1 and E_2:

$$y - x > \max\{ \text{MIN}, 2(1-\lambda_2)^{-1} \underline{\text{MIN}}, \gamma_1 x \} \qquad (3.7.2.22)$$

where

$$\gamma_1 \equiv \max\left\{ \frac{2\lambda_1}{(1-\lambda_2)(1-\lambda_1)^2}, \frac{2\mu_1}{2-(1+\mu_2)(1+\mu_1)^2} \right\}$$

For E_3 and E_4:

$$y - x > \max\{ \text{MIN}, 2(1-\lambda_2)^{-1} \underline{\text{MIN}}, \gamma_2 x \} \qquad (3.7.2.23)$$

where

$$\gamma_2 \equiv \max\left\{ \frac{2\lambda_2}{(1-\lambda_2)[2-(1+\mu_2)(1+\mu_1)]}, \frac{2\mu_2}{(1-\lambda_2)(1-\lambda_1)(1+\mu_2)-2\mu_2} \right\}$$

For E_5:

$$x \geq 2\underline{\text{MIN}} \quad \& \quad y - x > \gamma_3 x \qquad (3.7.2.24)$$

where

$$\gamma_3 \equiv \max\left\{ \frac{2\lambda_1(2-\lambda_1)}{(1-\lambda_1)^2}, \frac{2\mu_1(2+\mu_1)}{2-(1+\mu_1)^2} \right\}$$

For E_6:

$$x + y \leq \overline{\text{MAX}} \quad \& \quad y - x > \gamma_3 x \qquad (3.7.2.25)$$

From (3.7.2.22) - (3.7.2.25) and (3.7.2.7) we have $y - x > x \geq \text{MIN}$. Thus, by (3.7.2.7) and axioms F8, F12, and F15,

$$(y-x)(1+\mu_2) \geq y \ominus x$$

Since by (3.7.2.5), $y \ominus x > \varepsilon$, it follows that

$$y - x > \varepsilon(1+\mu_2)^{-1}$$

Therefore, by (3.7.2.6) and (3.7.2.22) - (3.7.2.25), termination is guaranteed if ε, x_0, and y_0 are chosen as follows.

For E_1 and E_2:

$$\varepsilon \geq (1+\mu_2) \max\{\mathrm{MIN},\ 2(1-\lambda_2)^{-1}\underline{\mathrm{MIN}},\ \gamma_1 y_0\}$$

For E_3 and E_4:

$$\varepsilon \geq (1+\mu_2) \max\{\mathrm{MIN},\ 2(1-\lambda_2)^{-1}\underline{\mathrm{MIN}},\ \gamma_2 y_0\}$$

For E_5:

$$\varepsilon \geq (1+\mu_2)\, \gamma_3 y_0 \quad \& \quad x_0 \geq 2\,\underline{\mathrm{MIN}}$$

For E_6:

$$\varepsilon \geq (1+\mu_2)\, \gamma_3 y_0 \quad \& \quad y_0 \leq \tfrac{1}{2}\overline{\mathrm{MAX}}$$

QED

Values of λ_i, μ_i, γ_i, MIN, and MAX are summarized below for the IBM 360/370, DEC PDP-11, and CDC 6000/Cyber-70 computers. To four decimal-place accuracy, $\underline{\mathrm{MIN}}$ = MIN and $\overline{\mathrm{MAX}}$ = MAX.

	IBM-S (chopped)	IBM-L (chopped)	DEC-S (rounded)	DEC-L (rounded)	CDC-S (chopped)
λ_1	$9.537_{10}{-7}$	$2.220_{10}{-16}$	$5.960_{10}{-8}$	$1.388_{10}{-17}$	$7.105_{10}{-15}$
λ_2	$8.941_{10}{-7}$	$2.082_{10}{-16}$	$5.960_{10}{-8}$	$1.388_{10}{-17}$	$7.105_{10}{-15}$
μ_1	0	0	$5.960_{10}{-8}$	$1.388_{10}{-17}$	0
μ_2	$5.960_{10}{-8}$	$1.388_{10}{-17}$	$5.960_{10}{-8}$	$1.388_{10}{-17}$	$2.524_{10}{-29}$
MIN	$5.398_{10}{-79}$	$5.398_{10}{-79}$	$2.939_{10}{-39}$	$2.939_{10}{-39}$	$7.829_{10}{-295}$
MAX	$7.237_{10}{+75}$	$7.237_{10}{+75}$	$1.701_{10}{+38}$	$1.701_{10}{+38}$	$2.530_{10}{+322}$
γ_1	$1.907_{10}{-6}$	$4.440_{10}{-16}$	$1.192_{10}{-7}$	$2.776_{10}{-17}$	$1.421_{10}{-14}$
γ_2	$1.788_{10}{-6}$	$4.164_{10}{-16}$	$1.192_{10}{-7}$	$2.776_{10}{-17}$	$1.421_{10}{-14}$
γ_3	$3.815_{10}{-6}$	$8.880_{10}{-16}$	$2.384_{10}{-7}$	$5.552_{10}{-17}$	$2.842_{10}{-14}$

Table 3.7.2.1. Values of λ_i, μ_i, γ_i, MIN, and MAX for the IBM 360/370, DEC PDP-11, and CDC 6000/Cyber-70 computers. (The suffixes "S" and "L" denote short and long precision, respectively. Also, "$d.ddd_{10}ee$" denotes $d.ddd \times 10^{ee}$.)

From table 3.7.2.1, several observations are made. (Recall that β = base of the arithmetic, t = number of digits in the fraction, and g = number of guard digits employed.) First, we note that the conventional bounds given by

$$\beta^{1-t} \quad \text{for chopped arithmetic}$$
$$\frac{1}{2}\beta^{1-t} \quad \text{for rounded arithmetic}$$

equal $\max\{\lambda_i, \mu_i\}$ to within (at least) four decimal-place accuracy, for each of the given computers. However, the table shows that these bounds are inadequate to capture the fact that with chopped arithmetic, $\lambda_2 < \lambda_1$ if $g < t$; $\mu_1 = 0$; and μ_2 is a factor of β^{-g} smaller than λ_1.

Also, to within (at least) four decimal-place accuracy,

$$\gamma_1 = 2\lambda_1, \quad \gamma_2 = 2\lambda_2, \quad \gamma_3 = 4\lambda_1$$

Consequently, for each of the given computers, E_1 and E_2 can be expected to be about twice as accurate as E_5 and E_6. E_3 and E_4 are expected to be marginally more accurate than E_1 and E_2 on the IBM 360/370 with either short or long precision, and not significantly more or less accurate on the other two computers.

Since E_5 requires $x_0 \geq 2\,\underline{\text{MIN}}$, and E_6 requires $y_0 \leq \frac{1}{2}\overline{\text{MAX}}$, we see that in addition to being more accurate, $E_1 - E_4$ can be guaranteed to work for a wider range of values than either E_5 or E_6.

Finally, for completeness, we mention that to within (at least) four decimal-place accuracy, Dekker's [12] bound for E_1 is identical to our $2\lambda_1$. However, without giving a derivation, he concludes that a bound of approximately $3\lambda_1$, rather than our $4\lambda_1$, is sufficient to guarantee that a similar program with E_6 for computing the midpoint of x and y will terminate.

Reasoning About Correctly Solving the Given Problem:

From the invariance of P, we are guaranteed that [x, y] contains a root of f. We now give conditions characterizing η, such that $y - x \leq \eta$.

If $y - x \geq \underline{MIN}$ upon termination, then from (3.7.2.7) and axioms F8, F12, and F15,

$$(y - x)(1 - \lambda_2) \leq y \ominus x$$

Since upon termination, $y \ominus x \leq \varepsilon$, it follows that

$$y - x \leq \max\{\underline{MIN}, \varepsilon(1 - \lambda_2)^{-1}\}$$

Because $\underline{MIN} \leq \varepsilon$, the desired result follows, provided η is chosen so that

$$\eta \geq \varepsilon(1 - \lambda_2)^{-1}$$

QED

Further Considerations:

For the sake of simplicity, the given program was developed to find only positive roots, α, of f. In fact, this restriction is not necessary for the preceding analysis to apply. Suppose f is defined on the intervals $[-a, -b]$, $[0]$, and $[c, d]$, where $a, b, c, d \in \mathbf{F}$, and $-a \leq -b \leq 0 \leq c \leq d$. By evaluating $f(-b)$, $f(0)$, and $f(c)$, it can immediately be determined whether either of the intervals $[-b, 0]$ or $[0, c]$ contain a root of f. For the interval $[c, d]$, the analysis given above applies. For the interval $[-a, -b]$, this analysis will also apply, provided x_0 and y_0 are chosen so that $x_0, y_0 \geq$ MIN, and $f(-E_j)$ is evaluated each iteration to determine which of x and y is to be updated.

Also, the analysis of the given program was performed assuming that f is computed exactly. If, in fact, f is computed with fp arithmetic, then the preceding analysis indicates that the program will converge to within η of a genuine root of some perturbed continuous function $f + \delta f$. Let α denote a real root of f, and let ξ denote the corresponding root of $f + \delta f$. Also, let $\underline{\mathrm{int}}(\alpha, \xi)$ denote the smallest closed interval containing both α and ξ. Wilkinson [70] has shown that under certain conditions,

$$[x, y] \cap \underline{\mathrm{int}}(\alpha, \xi) \neq \{\}$$

will be left invariant by a program similar to the one given above. We refer the interested reader to the reference for details.

3.8 Summary of the Approaches for Proving Program Correctness

> Quod nunc ratio est, impetus ante fuit.
> (What now is reason was formerly impulse.)
> -- Ovid, Remediorum Amoris

We have presented two approaches for reasoning about the correctness of floating-point (fp) programs. The first approach associates a set of fp numbers with each fp expression and models the assignment statement as a nondeterministic selector of one of the elements in the set.

This approach appears to adequately capture the fact that the result of an fp operation is an fp number chosen from an interval, and that the identity of the number is known with no more precision than knowledge of the endpoints of the interval will allow. However, since this approach does not capture the single-valuedness of the fp operations, it is inadequate to demonstrate certain significant properties of fp programs.

The second approach models the fp operations by single-valued functions, whose significant properties are characterized by 15 fp axioms. This set of axioms was noted to be sufficiently general to apply across a wide range of fp implementations: the axioms support various treatments of overflow and underflow, and either rounded or chopped arithmetic, provided that at least one guard digit is used. Also, these axioms were shown to be sufficiently powerful to prove several properties that have been reported to be desirable for fp arithmetic. It is believed that these axioms are the first attempt to model all the fp operations in terms of exactly two cropping functions.

This approach appears to provide an adequate framework for reasoning about the correctness of fp programs, systematically accounting for the limitations of range and precision embodied by these programs. Two ways were shown for specifying the invariant predicate associated with a loop in fp programs. One way is to make use of sequences, defined recursively in terms of fp operations. This approach frees invariance proofs from any consideration of error in the fp operations. In fact, with this approach, an invariance proof demonstrates only that the fp operations are applied by the program under the conditions and in the order dictated by the statically specified sequences. Thus, the invariance proof is used to determine whether the fp algorithm embodied by these sequences is _correctly implemented_. However, we saw that such an approach requires an error analysis to be performed inductively upon these sequences, when the inexactness of the fp operations must finally be taken into account, to reason about whether the implemented algorithm _solves a given problem_.

The alternative approach was to specify the invariant, not in terms of sequences, but only in terms of the initial and current values of the program variables. With this approach, the invariance proof is more complicated, because it must take into account the inexactness of fp arithmetic. However, a separate induction during the error analysis is not needed.

The former approach for specifying invariants provides a dramatic separation of concerns that is probably preferable for large programs. The latter approach, which consolidates some of these concerns and treats them together, may be preferred for programs that are sufficiently small, so that the amount of detail that must be handled at once remains manageable.

Finally, we showed that the common practice of modelling the fp operations by a single function that crops the result of the corresponding exact operation is invalid for many fp implementations. For schemes S_1 and S_3, we showed that _two_ cropping functions are necessary if and only if $g \leq t$. For scheme S_2, we showed that _two_ cropping functions are necessary if and only if $\beta = 2$ & $g < t$ or $\beta > 2$ & $g \leq t$.

Appendix

Some Theorems Provable from Axioms F1 - F15

> Numerical subroutines should deliver results which satisfy simple, useful mathematical laws whenever possible. [Such laws make] ... a great deal of difference between whether mathematical analysis of computational algorithms is worth doing or worth avoiding! Without any underlying symmetry properties, the job of proving interesting results becomes extremely unpleasant. The enjoyment of the tools one works with is, of course, an essential ingredient of successful work.
>
> -- Donald Knuth [42]

The index i is assumed to range over the set { 0, 1 }.

Theorems Characterizing the FP Parameters

TH1. $(\forall x \in \mathbb{F} : \; CR_1(x) = CR_2(x) = x)$

<u>Proof</u>:

$x \in \mathbb{F} \;\Rightarrow\; x \in \mathbb{R}$ — by F1

$\Rightarrow\; x \le CR_i(x) \le x$ — by F5

<u>QED</u>

TH2. $(\forall x \in \mathbb{R} : \; x \ge MAX \;\Rightarrow\; CR_1(x) = CR_2(x) = MAX)$

<u>Proof</u>:

$x \in \mathbb{R} \;\Rightarrow\; CR_i(x) \in \mathbb{F}$ — by F5

$\Rightarrow\; CR_i(x) \le MAX$ — by F3

$MAX \le x \;\;\&\;\; x \in \mathbb{R}$

$\Rightarrow\; CR_i(MAX) \le CR_i(x)$ — by F9

$\Rightarrow\; MAX \le CR_i(x)$ — by F3, TH1

<u>QED</u>

TH2.5 ($\forall x \in \mathbb{R}$: $\underline{\mathrm{MIN}} \leq x \leq \mathrm{MIN} \Rightarrow CR_1(x) = CR_2(x) = \mathrm{MIN}$)

<u>Proof</u>:

$x \in \mathbb{R}$ & $\underline{\mathrm{MIN}} \leq x \leq \mathrm{MIN}$

$\Rightarrow x > 0$ & $CR_i(x) \geq x(1 - \lambda_i)$ by F3, F4, F6, F8

$\Rightarrow CR_i(x) > 0$ by F7

$\Rightarrow \mathrm{MIN} \leq CR_i(x) \leq \mathrm{MIN}$ by F4, F5

<u>QED</u>

TH3. ($\forall x \in \mathbb{F}$: $-x \in \mathbb{F}$)

<u>Proof</u>:

$x \in \mathbb{F} \Rightarrow CR_i(-x) = -CR_i(x)$ by F10

$\Rightarrow CR_i(-x) = -x$ by TH1

$\Rightarrow -x \in \mathbb{F}$ by F5

<u>QED</u>

TH4. ($\forall x \in \mathbb{F}$: $x \geq -\mathrm{MAX}$)

<u>Proof</u>:

Obvious, by F3 and TH3. <u>QED</u>

TH5. ($\forall x \in \mathbb{F}$: $x < 0 \Rightarrow x \leq -\mathrm{MIN}$)

<u>Proof</u>:

Obvious, by F4 and TH3. <u>QED</u>

TH6. ($\forall x \in \mathbb{R}$: [$-\overline{\mathrm{MAX}} \leq x \leq -\underline{\mathrm{MIN}}$

$\Rightarrow x(1 + \mu_i) \leq CR_i(x) \leq x(1 - \lambda_i)$])

<u>Proof</u>:

Obvious, by F8, F10, and TH3. <u>QED</u>

The theorems which follow assume $\mathbb{D}$om = true. TH7 - TH27 were posed as desirable axioms and theorems for fp arithmetic by Knuth [42].

Theorems Characterizing FP Addition and Subtraction

TH7. $(\forall x, y \in \mathbb{F} : \; x \oplus y = y \oplus x)$

Proof:

$$x, y \in \mathbb{F} \;\Rightarrow\; [\, x \oplus y = CR_1(x+y) = y \oplus x \,] \;\vee$$

$$[\, x \oplus y = CR_2(x+y) = y \oplus x \,] \qquad \text{by F11, F12}$$

QED

TH8. $(\forall x, y \in \mathbb{F} : \; x \ominus y = x \oplus (-y))$

Proof:

Axiom F15. QED

TH9. $(\forall x, y \in \mathbb{F} : \; x \ominus y = -(y \ominus x))$

Proof:

$$\begin{aligned} x, y \in \mathbb{F} \;\Rightarrow\; x \ominus y &= x \oplus (-y) && \text{by F15} \\ &= CR_j(x-y) \quad \text{for } j=1 \vee j=2 && \text{by F11, F12} \\ &= -CR_j(y-x) && \text{by F10} \\ &= -(y \oplus (-x)) && \text{by F11, F12} \\ &= -(y \ominus x) && \text{by F15} \end{aligned}$$

QED

TH10. $(\forall x, y \in \mathbb{F} : \; -(x \oplus y) = (-x) \oplus (-y))$

Proof:

$$\begin{aligned} x, y \in \mathbb{F} \;\Rightarrow\; -(x \oplus y) &= -(x \ominus (-y)) && \text{by F15} \\ &= (-y) \ominus x && \text{by TH9} \\ &= (-y) \oplus (-x) && \text{by F15} \\ &= (-x) \oplus (-y) && \text{by TH7} \end{aligned}$$

QED

TH11. $(\forall x, y \in \mathbb{F} : \; y = -x \;\Rightarrow\; x \oplus y = 0)$

<u>Proof</u>:

$$\begin{aligned} x, y \in \mathbb{F} \;\&\; y = -x \;\Rightarrow\; x \oplus y &= CR_2(x+y) && \text{by F12} \\ &= CR_2(0) \\ &= 0 && \text{by F2, TH1} \end{aligned}$$

<u>QED</u>

Note: The implication in the other direction, required by Knuth, does not hold if for some $x, y \in \mathbb{F} : \; 0 < x + y < \underline{MIN} \;\Rightarrow\; CR_2(x+y) = 0$.

TH12. $(\forall x \in \mathbb{F} : \; x \oplus 0 = x)$

<u>Proof</u>:

$$\begin{aligned} x \in \mathbb{F} &\;\Rightarrow\; CR_1(x) = x \oplus 0 = CR_2(x) && \text{by F11, F12} \\ &\;\Rightarrow\; x \oplus 0 = x && \text{by TH1} \end{aligned}$$

<u>QED</u>

TH13. $(\forall x \in \mathbb{F} : \; 0 \ominus (0 \ominus x) = x)$

<u>Proof</u>:

$$\begin{aligned} x \in \mathbb{F} \;\Rightarrow\; 0 \ominus (0 \ominus x) &= 0 \oplus (-(0 \ominus x)) && \text{by F15} \\ &= 0 \oplus (-(0 \oplus (-x))) && \text{by F15} \\ &= -((-x) \oplus 0)) \oplus 0 && \text{by TH7 (twice)} \\ &= x && \text{by TH12 (twice)} \end{aligned}$$

<u>QED</u>

TH14. $(\forall x, y, z \in \mathbb{F} : \; x \leq y \;\Rightarrow\; x \oplus z \leq y \oplus z)$

<u>Proof</u>:

$$\begin{aligned} & x, y, z \in \mathbb{F} \;\&\; xy \geq 0 \\ &\Rightarrow [\, x \oplus z = CR_1(x+z) \;\&\; y \oplus z = CR_1(y+z) \,] \;\vee \\ &\quad [\, x \oplus z = CR_2(x+z) \;\&\; y \oplus z = CR_2(y+z) \,] \qquad \text{by F11, F12} \end{aligned}$$

$\Rightarrow \quad x \oplus z \leq y \oplus z$ by F9, $x \leq y$

$x, y, z \in \mathbb{F}$ & $xy \leq 0$ & $x \leq y$

$\Rightarrow$ [$x \leq 0 \leq y$ & $z \leq 0$] $\vee$

[$x \leq 0 \leq y$ & $z \geq 0$]

$\Rightarrow$ [$x \oplus z = CR_1(x+z) \leq CR_1(z) = z$ &

$y \oplus z = CR_2(y+z) \geq CR_2(z) = z$] $\vee$

[$x \oplus z = CR_2(x+z) \leq CR_2(z) = z$ &

$y \oplus z = CR_1(y+z) \geq CR_1(z) = z$] by F11, F12, TH1

$\Rightarrow$ $x \oplus z \leq y \oplus z$

<u>QED</u>

TH15. ($\forall w, x, y, z \in \mathbb{F}$: $w \geq 0$ & $z \geq 0$ $\Rightarrow$ $x \oplus y \leq (x \oplus w) \oplus (y \oplus z)$)

<u>Proof</u>:

$w, x \in \mathbb{F}$ & $w \geq 0$

$\Rightarrow$ $0 \oplus x \leq w \oplus x$ by TH14

$\Rightarrow$ $x \oplus 0 \leq x \oplus w$ by TH7

$\Rightarrow$ $x \leq x \oplus w$ by TH12

$w, x, y \in \mathbb{F}$ & $w \geq 0$ & $z \geq 0$

$\Rightarrow$ $x \oplus y \leq (x \oplus w) \oplus y$ by TH14, $x \leq x \oplus w$

$= y \oplus (x \oplus w)$ by TH7

$\leq (y \oplus z) \oplus (x \oplus w)$ by TH14, $y \leq y \oplus z$

$= (x \oplus w) \oplus (y \oplus z)$ by TH7

<u>QED</u>

Theorems Characterizing FP Multiplication and Division

TH16. ($\forall x, y \in \mathbb{F}$: $x \otimes y = y \otimes x$)

<u>Proof</u>:

$$x, y \in \mathbb{F} \Rightarrow x \circledast y = CR_1(x\,y) = y \circledast x$$ by F13

QED

TH17. $(\forall x, y \in \mathbb{F} : (-x) \circledast y = -(x \circledast y))$

<u>Proof</u>:

$$x, y \in \mathbb{F} \Rightarrow -x \in \mathbb{F} \ \& \ (-x) \circledast y = CR_1(-x\,y)$$ by TH3, F13

$$= -CR_1(x\,y)$$ by F10

$$= -(x \circledast y)$$ by F13

QED

TH18. $(\forall x \in \mathbb{F} : 1 \circledast x = x)$

<u>Proof</u>:

$$x \in \mathbb{F} \Rightarrow 1 \circledast x = CR_1(x)$$ by F13

$$= x$$ by TH1

QED

The proofs of TH19 - TH24 are very similar to the proofs of TH16 - TH18, and are omitted.

TH19. $(\forall x, y \in \mathbb{F} : (x = 0 \vee y = 0) \Rightarrow x \circledast y = 0)$

Note: The implication in the other direction, required by Knuth, does not hold if for some $x, y \in \mathbb{F} : 0 < x\,y < \underline{MIN} \Rightarrow CR_1(x\,y) = 0$.

TH20. $(\forall x, y \in \mathbb{F} : (-x) \oslash y = x \oslash (-y))$

TH21. $(\forall x, y \in \mathbb{F} : (-x) \oslash y = -(x \oslash y))$

TH22. $(\forall x \in \mathbb{F} : 0 \oslash x = 0)$

TH23. $(\forall x \in \mathbb{F} : \; x \oslash 1 = x)$

TH24. $(\forall x \in \mathbb{F} : \; x \oslash x = 1)$

TH25. $(\forall x, y, z \in \mathbb{F} : \; x \le y \;\&\; z > 0 \;\Rightarrow\; x \otimes z \le y \otimes z)$

Proof:

$x, y, z \in \mathbb{F} \;\&\; x \le y \;\&\; z > 0$

$\Rightarrow \; x\,z \le y\,z$

$\Rightarrow \; CR_1(x\,z) \le CR_1(y\,z)$ by F9

$\Rightarrow \; x \otimes z \le y \otimes z$ by F13

QED

TH26. $(\forall x, y, z \in \mathbb{F} : \; x \le y \;\&\; z > 0 \;\Rightarrow\; x \oslash z \le y \oslash z)$

Proof:

Similar to the proof of TH25. QED

TH27. $(\forall x, y, z \in \mathbb{F} : \; x \le y \;\&\; z > 0 \;\Rightarrow\; z \oslash x \ge z \oslash y)$

Proof:

Similar to the proof of TH25. QED

Theorems Characterizing Other Useful FP Properties

TH28. $(\forall x, y \in \mathbb{F} : \; x \le y \;\Rightarrow\; y \ominus x \ge 0)$

Proof:

$x, y \in \mathbb{F} \;\Rightarrow\; y \ominus x = y \oplus (-x)$ by F15

$= CR_j(y - x)$ for $j = 1 \vee j = 2$ by F11, F12

$\ge CR_j(0)$ by F9, $x \le y$

$= 0$ by F2, TH1

QED

TH29. $(\forall x, y \in \mathbb{F} : (x \geq 0 \;\&\; y \geq 1) \Rightarrow x \circledast y \geq x)$

<u>Proof</u>:

$$x, y \in \mathbb{F} \Rightarrow x \circledast y = CR_1(x\,y) \qquad \text{by F13}$$
$$\geq CR_1(x) \qquad \text{by F9, } x \geq 0,\ y \geq 1$$
$$= x \qquad \text{by TH1}$$

<u>QED</u>

The proofs of TH30 - TH35 are very similar to the proofs of TH28 - TH29, and are omitted.

TH30. $(\forall x, y \in \mathbb{F} : (x \geq 0 \;\&\; 0 \leq y \leq 1) \Rightarrow 0 \leq x \circledast y \leq x)$

TH31. $(\forall x, y \in \mathbb{F} : (x \geq 0 \;\&\; y \geq 1) \Rightarrow 0 \leq x \oslash y \leq x)$

TH32. $(\forall x, y \in \mathbb{F} : (x \geq 0 \;\&\; 0 < y \leq 1) \Rightarrow x \oslash y \geq x)$

TH33. $(\forall x, y \in \mathbb{F} : 0 < x \leq y \Rightarrow 0 \leq x \oslash y \leq 1)$

TH34. $(\forall x \in \mathbb{F} : x \circledast x \geq 0)$

Note: $x \neq 0 \Rightarrow x \circledast x > 0$ is false if for some $x \in \mathbb{F} : 0 < x^2 < \underline{MIN}$ $\Rightarrow CR_1(x^2) = 0$.

TH35. $(\forall x \in \mathbb{F} : x > 0 \Rightarrow 1 \oslash x \geq 0)$

Note: $x \neq 0 \Rightarrow 1 \oslash x \neq 0$ is false if for some $x \in \mathbb{F} : 0 < 1/x < \underline{MIN}$ $\Rightarrow CR_1(1/x) = 0$.

TH36. $(\forall x, y \in \mathbb{F} : y \geq 0 \Rightarrow x \circledast y \geq x)$

<u>Proof</u>:

$x, y \in \mathbb{F}$ & $y \geq 0$

$\Rightarrow$ $0 \circledast x \leq y \circledast x$ by TH14

$\Rightarrow$ $x \circledast 0 \leq x \circledast y$ by TH7

$\Rightarrow$ $x \leq x \circledast y$ by TH12

QED

TH37. ($\forall w, x, y, z \in \mathbb{F}$: $xy \leq wz \Rightarrow x \circledast y \leq w \circledast z$)

Proof:

$w, x, y, z \in \mathbb{F} \Rightarrow$ [$x \circledast y$ $CR_1(xy)$ &

$w \circledast z = CR_1(wz)$] by F13

$\Rightarrow$ $x \circledast y \leq w \circledast z$ by F9, $xy \leq wz$

QED

TH38. ($\forall w, x, y, z \in \mathbb{F}$: $x/y \leq w/z \Rightarrow x \oslash y \leq w \oslash z$)

Proof:

Similar to the proof of TH37. QED

TH39. ($\forall w, x, y, z$ $\mathbb{F}$: [$(xy \geq 0 \Leftrightarrow wz \geq 0)$ $\vee$ $x+y \in \mathbb{F}$ $\vee$ $w+z \in \mathbb{F}$]

$\Rightarrow$ [$x+y \leq w+z \Rightarrow x \circledast y \leq w \circledast z$])

Proof:

[$w, x, y, z \in \mathbb{F}$ & $xy \geq 0 \Leftrightarrow wz \geq 0$]

$\Rightarrow$ [$x \circledast y = CR_j(x+y)$ &

$w \circledast z = CR_j(w+z)$] for $j = 1 \vee j = 2$ by F11, F12

$\Rightarrow$ $x \circledast y \leq w \circledast z$ provided $x+y \leq w+z$ by F9

$w, x, y, z \in \mathbb{F}$

$\Rightarrow$ $x \circledast y = CR_j(x+y)$ for $j = 1 \vee j = 2$ by F11, F12

$\leq CR_j(w+z)$ provided $x+y \leq w+z$ by F9

$= w+z$ by TH1, $w+z \in \mathbb{F}$

$= CR_j(w+z) \quad \text{for } j=1 \ \& \ j=2$ by TH1

$= w \oplus z$ by F11, F12

The proof is similar when $x+y \in \mathbb{F}$.

QED

TH40. $(\forall x, y \in \mathbb{F} : [\, x = y \ \vee \ MIN \le y-x \le MAX \,]$

$\Rightarrow \quad x \le x \oplus ((y \ominus x) \oslash 2) \le y \quad)$

Proof:

$x, y \in \mathbb{F} \ \& \ x = y$

$\Rightarrow \ y \ominus x = y \oplus (-x) = 0$ by F15, TH11

$\Rightarrow \ (y \ominus x) \oslash 2 = 0$ by TH22

$\Rightarrow \ x \oplus ((y \ominus x) \oslash 2) = x$ by TH12

$\Rightarrow \ x = x \oplus ((y \ominus x) \oslash 2) = y$

$x, y \in \mathbb{F} \ \& \ MIN \le y-x$

$\Rightarrow \ y > x$ by F6

$\Rightarrow \ y \ominus x \ge 0$ by TH28

$\Rightarrow \ (y \ominus x) \oslash 2 \ge 0$ by TH31

$\Rightarrow \ x \oplus ((y \ominus x) \oslash 2) \ge x$ by TH36

Let $\Delta y = y-x$ for $x, y \in \mathbb{F}$ & $MIN \le \Delta y \le MAX$.

We now show:

$$(y-\Delta y) \oplus [\, (y \ominus (y-\Delta y)) \oslash 2 \,] \le y$$

Let $\overline{\Delta y} \equiv \min\{ z \in \mathbb{F} : 0 < \Delta y \le z \}$.

Now, $\overline{\Delta y} \oslash 2 = CR_1(\frac{1}{2}\overline{\Delta y})$ by F14

Suppose $\Delta y \le \overline{\Delta y} \le 2\,MIN$. Then,

$CR_1(\frac{1}{2}\overline{\Delta y}) \le MIN$ by F5

$\leq \; \Delta y$ by hypothesis

Suppose $2\,MIN \leq \overline{\Delta y} \leq MAX$. Then,

$$CR_1(\tfrac{1}{2}\,\overline{\Delta y}) \;\leq\; \tfrac{1}{2}\,\overline{\Delta y}\,(1+\mu_1) \qquad \text{by F8}$$

$$< \; \overline{\Delta y} \qquad \text{by F7, } \mu_1 < 1$$

By definition of $\overline{\Delta y}$,

$$\overline{\Delta y} \;>\; \overline{\Delta y} \oslash 2 \in \mathbb{F} \;\Rightarrow\; \Delta y \geq \overline{\Delta y} \oslash 2$$

Thus, $MIN \leq \Delta y \leq MAX \;\Rightarrow\; \Delta y \;\geq\; \overline{\Delta y} \oslash 2.$ (3.7.2.1)

$$y \ominus (y - \Delta y) \;=\; CR_j(\Delta y) \qquad \text{for } j = 1 \vee j = 2 \qquad \text{by F11, F12, F15}$$

$$\leq \; \overline{\Delta y} \qquad \text{by F5, } \Delta y \leq \overline{\Delta y} \in \mathbb{F}$$

$$\Rightarrow \; (y \ominus (y - \Delta y)) \oslash 2 \;\leq\; \overline{\Delta y} \oslash 2 \qquad \text{by TH26}$$

$$\leq \; \Delta y \qquad \text{by (3.7.2.1)}$$

$$\Rightarrow \; (y - \Delta y) \oplus (\,(y \ominus (y - \Delta y)) \oslash 2$$

$$\leq \; (y - \Delta y) \oplus \Delta y \qquad \text{by TH7, TH14}$$

$$= \; CR_k(y) \qquad \text{for } k = 1 \vee k = 2 \qquad \text{by F11, F12}$$

$$= \; y \qquad \text{by TH1}$$

<u>QED</u>

Bibliography

Note: Items indicated by "*" are not referenced in the text.

[1] Aggarwal, V.B. and Burgmeier, J.W. A Roundoff Model with Applications to Arithmetic Expressions. SIAM J. Comput. 8, 1 (Feb. 1979), 60-72.

[2] Anderson, S.F., Earle, J.G., Goldschmidt, R.E., and Powers, D.M. The IBM System/360 Model 91: Floating-Point Execution Unit. [illegible] J., Jan. 1967.

[3] A Proposed Standard for Binary Floating-Point Arithmetic. SIGNUM Newsletter, Oct. 1979.

[4] Brent, R.P. On the Precision Attainable with Various Floating-Point Number Systems. IEEE Trans. Comput. C-22, 6 (June 1973), 601-607.

[5] Brown, W.S. and Richman, P.L. The Choice of Base. Comm. ACM 12, 10 (Oct. 1969), 560-561.

[6] Brown, W.S. A Realistic Model of Floating-Point Computation. Proc. Symp. Math. Research Cntr., U. Wisconsin, Madison, March 1977. In Math. [illegible] III, J.R. Rice, Ed., Academic Press, New York, 1977, pp. 343-360.

[7] Bustoz, J., Feldstein, A., and Goodman, R. Improved Trailing Digits Estimates Applied to Optimal Computer Arithmetic. J. ACM 26, 4 (Oct. 1979), 716-730.

[8] Cody, W.J. Jr., Static and Dynamic Numerical Characteristics of Floating-Point Arithmetic. IEEE Trans. Comput. C-22, 6 (June 1973), 598-601.

[9] Coonen, J.T. An Implementation Guide to a Proposed Standard for Floating-Point Arithmetic. [illegible] 13, 1 (Jan. 1980), 68-79.

[10] Dahlquist, G. and Björck, A. Numerical Methods. N. Anderson, Tr., Prentice-Hall, Englewood Cliffs, NJ, 1974.

[11] Dekker, T.J. A Floating-Point Technique for Extending the Available Precision. Numer. Math. 18 (1971), 224-242.

[12] Dekker, T.J. Correctness Proofs and Machine Arithmetic. Proc. IFIP TC2 Working Conf. Performance Evaluation Numer. Software, Baden bei Wien, Austria, Dec. 1978.

[13] De Millo, R.A., Lipton, R.J., and Perlis, A.J. Social Processes and Proofs of Theorems and Programs. Comm. ACM 22, 5 (May 1979), 271-280.

[14] De Millo, R.A., Lipton, R.J., and Perlis, A.J. Response to Dijkstra's commentary on "Social Processes and Proofs of Theorems and Programs". ACM SIGSOFT, Software Engineering Notes 3, 2 (April 1978), 16-17.

[15] Dijkstra, E.W. Guarded Commands, Nondeterminacy, and Formal Derivation of Programs. Comm. ACM 18, 8 (Aug. 1975), 453-457.

[16] Dijkstra, E.W. Discipline of Programming. Prentice-Hall, Englewood Cliffs, NJ, 1976.

[17] *Fettweis, A. On Properties of Floating-Point Roundoff Noise. IEEE Trans. Acoustics, Speech, and Signal Process. ASSP-22, 11 (Apr. 1974), 149-151.

[18] Floyd. R.W. Assigning Meanings to Programs. Proc. Amer. Math. Soc. Symp. Applied Math. 19 (1967), 19-31.

[19] Forsythe, G.E., Malcolm, M.A., and Moler, C.B. Computer Methods for Mathematical Computations. Prentice-Hall, Englewood Cliffs, NJ, 1977.

[20] Garner, H.L. A Survey of Some Recent Contributions to Computer Arithmetic. IEEE Trans. Comput. C-25, 12 (Dec. 1976), 1277-1282.

[21] Garner, H.L. Theory of Computer Addition and Overflows. IEEE Trans. Comput. C-27, 4 (April 1978), 297-301.

[22] Goldstine, H.H. and von Neumann, J. Planning and Coding of Problems for an Electronic Computing Instrument. Report prepared for U.S. Army Ordinance Dept., part 2, vol. 1, 1947. Reprinted in John von Neumann, Collected Works, 5, A.H. Taub., Ed. The MacMillan Co., New York, pp. 80-151.

[23] Goodman, R. and Feldstein, A. Effect of Guard Digits and Normalization Options on Floating-Point Multiplication. Computing 18 (1977), 93-106.

[24] Gries, D. Describing an Algorithm by Hopcroft. Acta Infor. 2 (1973), 97-109.

[25] Gries, D. Proofs. In Programming Methodology: A Collection of Articles by Members of IFIP WG2.3, D. Gries, Ed., Springer-Verlag, New York, 1978, p. 75.

[26] Grishman, R. Assembly Language Programming for the Control Data 6000 Series and the Cyber 70 Series. Algorithmics Press, New York, 1974.

[27] Higbie, L.C. Vector Floating-Point Data Format. IEEE Trans. Comput. C-25, 1 (Jan. 1976), 25-32.

[28] Hill, F.J. and Peterson, G.R. Digital Systems: Hardware Organization and Design. John Wiley and Sons, Inc., New York, 1973.

[29] Hoare, C.A.R. An Axiomatic Basis for Computer Programming. Comm. ACM 12, 10 (Oct. 1969), 576-583.

[30] Hoare, C.A.R. Proof of a Program: FIND. Comm. ACM 14, 1 (Jan. 1971), 39-45.

[31] Hoare, C.A.R. Procedures and Parameters: An Axiomatic Approach. Proc. Symp. Semantics Algorithmic Langs., Lecture Notes in Math. 188, Springer-Verlag, New York, 1971, pp. 102-116.

[32] Hull, T.E., Enright, W.H., and Sedgwick, A.E. The Correctness of Numerical Algorithms. Proc. SIGPLAN Symp. Proofs of Assertions about Programs, Las Cruces NM, 1972, pp. 66-73.

[33] Hull, T.E. Desirable Floating-Point Arithmetic and Elementary Functions for Numerical Computation. Proc. 4th Symp. Comput. Arith., UCLA, Santa Monica, Ca., Oct. 1978, IEEE Comput. Soc., pp. 63-69.

[34] Hull, T.E. Correctness of Numerical Software. Proc. IFIP TC2 Working Conf. Performance Evaluation Numer. Software, Baden bei Wien, Austria, Dec. 1978.

[35] IBM System/360 Principles of Operation, GA22-6821-8. IBM Corp., Poughkeepsie, NY, 1970.

[36] IBM System/370 Principles of Operation, GA22-7000-3. IBM Corp., Poughkeepsie, NY, 1973.

[37] Jackson, K.R. Proving Properties about Subroutines that Solve ODE's Numerically. M.S. Thesis, U. of Toronto, 1974.

[38] Johnson, D.B., Miller, W., Minnihan, B., and Wrathall, C. Reducibility Among Floating-Point Graphs. J. ACM 26, 4 (Oct. 1979), 739-760.

[39] Kahan, W. A Survey of Error Analysis. Proc. IFIP Congress 71, Ljubljana, Yugoslavia, Aug. 1971, North-Holland, pp. 1214-1239.

[40] Kaneko, T. and Liu, B. On Local Roundoff Errors in Floating-Point Arithmetic. J. ACM 20, 3 (July 1973), 391-398.

[41] Kent, J.G. Highlights of a Study of Floating-Point Instructions. IEEE Trans. Comput. C-26, 7 (July 1977), 660-666.

[42] Knuth, D.E. The Art of Computer Programming, Vol. 2: Seminumerical Algorithms. Addison-Wesley, Reading, Ma., 1969.

[43] Kuck, D.J., Parker, Jr., D.S., and Sameh, A.H. Analysis of Rounding Methods in Floating-Point Arithmetic. IEEE Trans. Comput. C-26, 7 (July 1977), 643-650.

[44] Kuki, H. and Cody, W.J. A Statistical Study of the Accuracy of Floating-Point Number Systems. Comm. ACM 16, 4 (April 1973), 223-230.

[45] Kulisch, U. Mathematical Foundation of Computer Arithmetic. IEEE Trans. Comput. C-26, 7 (July 1977), 610-621.

[46] Kulisch, U. and Mira[illegible]er, W.L. Computer Arithmetic in Theory [illegible] Practice. To be published by Academic Press.

[47] Lamport, L. Contribution to ACM Forum: Comments on Social Processes and Proofs. Comm. ACM 22, 11 (Nov. 1979), 624.

[48] *Larson, J. and S[illegible]eh, A. Efficient Calculation of the Effects of Roundoff Errors. ACM Trans. Math. Software 4, 3 (Sept. 1978), 228-236.

[49] Liddiard, L.A. Required Scientific Floating-Point Arithmetic. Proc. 4th Symp. Comput. Arith., UCLA, Santa Monica, Ca., Oct. 1978, IEEE C[illegible]put. Soc., pp. 56-62.

[50] Metropolis, N. and Ashenhurst, R.L. Significant Digit Arithmetic. IRE Trans. Electron. Comput. EC-7 (1958), 265-267.

[51] Metropolis, N. Methods of Significance Arithmetic. Proc. State of the Art Numer. Analysis, U. of York, England, April 1976. In [illegible] State of [illegible] [illegible] in Numerical Analysis, D. Jacobs, Ed. Academic Press, New York, pp. 179-192.

[52] Miller, W. Graph Transformations for Roundoff Analysis. SIAM J. Comput. 5, 2 (June 1976), 204-216.

[53] Moore, R.E. [illegible] Analysis. Prentice-Hall, Englewood Cliffs, NJ, 1966.

[54] Nickel, K. Interval Analysis. Proc. State of the Art N[illegible]er. Analysis, U. of York, England, April 1976. In The State of the [illegible] in Numerical [illegible], D. Jacobs, Ed. Academic Press, New York, pp. 193-225.

[55] Olver, F.W.J. A New Approach to Error Arithmetic. SIAM J. [illegible]. Anal. 15, 2 (April 1978), 368-393.

[56] Owicki, S. and Gries, D. An Axiomatic Proof Technique for Parallel Programs. Acta Infor. 6 (1976), 319-340.

[57] [illegible] 11/60 Processor [illegible], EB-06498. Digital Equipment Corp., Maynard, Ma., 1977.

[58] PDP 11/04/34/45/55/60 Processor Handbook, EB-09430. Digital Equipment Corp., Maynard, Ma., 1978.

[59] Reinsch, C.H. Principles and Preferences for C[illegible]puter Arithmetic. SIGNUM Newsletter 14, 1 ([illegible]rch 1979), 12-27.

[60] Sanderson, J.G. Proof of Convergence for the Tridiagonal QL Algorithm in Floating-Point Arithmetic. Ph.D. Thesis, U. of New Mexico, 1977.

[61] *Sripad, A.B., Snyder, D.L. Quantization Errors in Floating-Point Arithmetic. IEEE Trans. Acoustics, Speech, and Signal Process. ASSP-26, 5 (Oct. 1978), 456-463.

[62] Sterbenz, P.H. Understandable Arithmetic. Proc. 3rd Symp. Comput. Arith., Southern Methodist U., Dallas, Tx., Nov. 1975, IEEE Comput. Soc., pp. 33-35.

[63] Thiran, J.P. Error Bounds for Floating-Point Addition Using Guard Digits. Proc. 1978 Internat. Symp. Circuits and Systems, New York, May 1978, IEEE, pp. 1034-1037.

[64] *Tsao, N.K. On the Distribution of Significant Digits and Roundoff Errors. Comm. ACM 17, 5 (May 1974), 269-271.

[65] Turing, A.M. Rounding-Off Errors in Matrix Processes. Quart. J. Mech. 1 (1948), 287-308.

[66] Turing, A.M. Checking a Large Routine. Proc. Conf. High Speed Automatic Calculating Machines, Cambridge U. Math. Lab., Cambridge, 1949, pp. 67-68.

[67] Virkkunen, V-E.J. A Unified Approach to Floating-Point Rounding with Applications to Multiple-Precision Summation. Report A-1980-1, Dept. of Comput. Sci., U. Helsinki, Finland, March 1980.

[68] von Neumann, J. and Goldstine, H.H. Numerical Inverting of Matrices of High Order. Bull. Amer. Math. Soc. 53 (1947), 1021-1099.

[69] Walker, B.J., Kemmerer, R.A., and Popek, G.J. Specification and Verification of the UCLA Unix Security Kernel. Comm. ACM 23, 2 (Feb. 1980), 118-131.

[70] Wilkinson, J.H. Rounding Errors in Algebraic Processes. Prentice-Hall, Englewood Cliffs, NJ, 1963.

[71] Wilkinson, J.H. The Algebraic Eigenvalue Problem. Clarendon Press, Oxford, 1965.

[72] Wirth, N. The Programming Language PASCAL. Acta. Infor. 1 (1971), 35-63.

[73] Wirth, N. Systematic Programming: An Introduction. Prentice-Hall, Englewood Cliffs, NJ, 1973.

[74] Yohe, J.M. Roundings in Floating-Point Arithmetic. IEEE Trans. Comput. C-22, 6 (June 1973), 577-586.