A FEASIBILITY ANALYSIS OF FIXED-SLASH RATIONAL ARITHMETIC*

Peter Kornerup
David W. Matula

Computer Science Department Aarhus University Aarhus, Denmark
Computer Science Department Southern Methodist University Dallas, Texas

Abstract

An investigation of the feasibility of a finite precision approximate rational arithmetic based on fixed-slash representation of rational numbers is presented. Worst-case and average-case complexity analyses of the involved rounding algorithm (an extended shift-subtract gcd algorithm) are presented. The results are applied to a proposed hardware realization of a fixed-slash arithmetic unit.

Key Words and Phrases:

Rational arithmetic, Fixed-slash arithmetic, Rounding, Finite precision, GCD Algorithm, Continued fractions, Convergents, Worst-case complexity analysis, Average-case complexity analysis, Arithmetic unit design.

CR Categories:

5.11, 6.32, 5.25, 3.15

I. Introduction and Summary

This paper presents some partial results of ongoing research on the foundation and implementation of a finite precision approximate rational arithmetic, based on the fixed- and floating-slash representations presented at the previous symposium on computer arithmetic.[1] Hitherto, floating-point approximations of real numbers have been considered the (almost only) viable number representation for scientific/numerical computations. The fixed- and floating-slash representations provide an alternative to floating-point representations, an alternative which has a number of merits worth investigating.

This presentation concentrates on the algorithmic and architectural feasibility of the involved arithmetic, primarily the rounding algorithm. Loosely speaking, this rounding algorithm is based on the computation of the "last representable" continued fraction approximation of the number to be rounded. This is certainly not a trivial rounding algorithm compared to roundings used in floating-point systems. The present analysis demonstrates that this rounding can be realized in today's technology at a speed which makes such approximate rational arithmetic feasible, although not fully competitive in 'very high speed' computers.

Complexity analysis and arithmetic unit design is pursued for the rounding associated with fixed-slash representation. The complexity analysis of the equivalent rounding procedure for floating-slash representation is beyond the scope of this paper, although additional research has shown comparable complexity results are obtainable.

II. Binary Fixed-Slash Systems

Since it is not the purpose of this paper to detail the formal aspects of approximate rational number systems, we only introduce the notations and definitions necessary for analysis of binary fixed-slash systems, which are the subject of this paper. An in-depth treatment covering more general finite precision rational number systems is under preparation.[2,3]

We may think of the binary fixed-slash representation of a rational number as a representation where in a computer word one bit is set aside for a sign, and two fields each of N bit width are used to represent respectively, the numerator and denominator of a fraction.

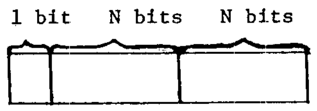

Figure 1.

The monotonically increasing sequence of such fixed-slash representable irreducible fractions between zero and unity is well known in classical number theory (e.g.[4]) as the Farey series F_{2^N-1}. As the other fixed-slash representable values are then simply determined by inverses and negation, the theory of Farey fractions provides a foundation for these number systems. For our purposes, we characterize this binary fixed-slash system by the set of ordered pairs (ignoring sign)

$$K = K(N) = \{(p,q) \mid 0 \leq p \leq 2^N-1,\ 0 \leq q \leq 2^N-1\}. \quad (1)$$

We shall henceforth refer to the fixed-slash system K with the dependence on N and addition of the sign to determine negative values implicitly understood. Note that K contains reducible as well as irreducible

*This research was supported in part by the National Science Foundation under Grant MCS 77-21510.

fractions, and every irreducible fraction $\frac{p}{q}$ for $(p,q) \in K$ is said to have multiplicity given by the maximum m such that $(mp,mq) \in K$.

Utilizing the notation $[a_0,a_1,\ldots]$ for the continued fraction

$$a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cdots}} \qquad a_i \geq 0 \tag{2}$$

where the quotients a_i are assumed to be integral, any non-negative rational number $\frac{p}{q}$ has a finite expansion

$$\frac{p}{q} = [a_0,a_1,\ldots,a_n]$$

which is unique (canonical) with the added requirements
$a_0 \geq 0$; $a_i \geq 1, i = 1,\ldots,n-1$; and $a_n \geq 2$ when $n \geq 1$.
The truncated continued fractions

$$\frac{p_i}{q_i} \equiv [a_0,a_1,\ldots,a_i],\ i = 0,1,\ldots,n \tag{3}$$

yield rational numbers which form a sequence of continued fraction approximations of $\frac{p}{q}$ (called the convergents) whose properties can be summarized as follows.

Theorem 1.

The convergents $\frac{p_i}{q_i} = [a_0,a_1,\ldots,a_i]$ of $\frac{p}{q} = [a_0,a_1,\ldots,a_n]$ for $i = 0,1,\ldots,n$ satisfy the following properties:

i) (recursive ancestry)

With $p_{-2} = 0, p_{-1} = 1, q_{-2} = 1$ and $q_{-1} = 0$,

$$p_i = a_i p_{i-1} + p_{i-2},$$
$$q_i = a_i q_{i-1} + q_{i-2},$$

ii) (irreducibility)

$$\gcd(p_i,q_i) = 1,$$

iii) (adjacency)

$$q_i p_{i-1} - p_i q_{i-1} = (-1)^i,$$

iv) (alternating convergence)

$$\frac{p_0}{q_0} < \frac{p_2}{q_2} < \ldots < \frac{p_{2j}}{q_{2j}} < \ldots \leq \frac{p}{q} \leq \ldots < \frac{p_{2j-1}}{q_{2j-1}} < \ldots < \frac{p_1}{q_1},$$

v) (best rational approximation)

$$\frac{r}{s} \neq \frac{p_i}{q_i},\ s \leq q_i \Longrightarrow \left| \frac{r}{s} - \frac{p}{q} \right| > \left| \frac{p_i}{q_i} - \frac{p}{q} \right|,$$

vi) (quadratic convergence)

$$\left| \frac{p_i}{q_i} - \frac{p}{q} \right| \leq \frac{1}{q_i q_{i+1}} \quad \text{for } i \leq n-1.$$

Proof: These are all classical results on continued fractions.[4] ▌

Since the quotients a_i of the expression $\frac{p}{q} = [a_0,a_1,\ldots,a_n]$ are the quotients obtained from the standard Euclidian GCD Algorithm applied to p,q, we may simply extend the Euclidian Algorithm by incorporating computation of p_i and q_i in parallel with computation of the a_i to achieve an algorithm for determination of all convergents of $\frac{p}{q}$.

Algorithm EC: (Euclidian-Convergent Algorithm)
For any $p \geq 0$, $q \geq 1$, let

$$b_{-2} = p;\ \ p_{-2} = 0;\ \ q_{-2} = 1;$$
$$b_{-1} = q;\ \ p_{-1} = 1;\ \ q_{-1} = 0.$$

Determine a_i as the quotient, and b_i as the non-negative remainder of the division of b_{i-2} by b_{i-1}, so

$$b_{i-2} = b_{i-1} \cdot a_i + b_i,$$

and compute

$$p_i = p_{i-1} \cdot a_i + p_{i-2},$$
$$q_i = q_{i-1} \cdot a_i + q_{i-2},$$

for $i = 0,1,\ldots$ until $i = n$, where $b_{n-1} \neq 0$, $b_n = 0$. ▌

For any rational $x = \frac{p}{q} = [a_0,a_1,\ldots,a_n]$ or irrational $x = [a_0,a_1,\ldots]$, the convergents $\frac{p_0}{q_0}, \frac{p_1}{q_1}, \frac{p_2}{q_2},\ldots$ represent successively finer approximations to x. For the fixed-slash system K, the Euclidian-Convergent Algorithm may be used to determine the K-finest convergent given by:

$$\Phi_K(x) = \begin{cases} \dfrac{p_n}{q_n} & \text{if } x = \dfrac{p}{q} = \dfrac{p_n}{q_n} \text{ with } (p_n,q_n) \in K, \\ \dfrac{p_i}{q_i} & \text{if } (p_i,q_i) \in K \text{ and } (p_{i+1},q_{i+1}) \notin K, \\ -\Phi_K(-x) & \text{if } x < 0. \end{cases} \tag{4}$$

The function Φ_K is now shown to be a "rounding" for the fixed-slash system K.

Theorem 2:

The function Φ_K given by (4) is a rounding scheme in the sense of Kulish[5], i.e.:

i) (monotonic)

$$x < y \Rightarrow \Phi_K(x) \leq \Phi_K(y),$$

ii) (antisymmetric)

$$\Phi_K(-x) = -\Phi_K(x),$$

iii) (fixed points)

$$x = \frac{p}{q},\ (p,q) \in K \Longrightarrow \Phi_K(x) = \frac{p}{q},$$

$$x = -\frac{p}{q},\ (p,q) \in K \Longrightarrow \Phi_K(x) = -\frac{p}{q}.$$

Furthermore:

iv) (exact inverses)

$$\Phi_K(x) = \frac{1}{\Phi_K(\frac{1}{x})}.$$

Proof: The properties follow readily from the definition (4) and the theory of continued fractions.[4] |

For each interval (x,x') between successive members of a finite precision number system a rounding yields a splitting point s with $x \leq s \leq x'$ such that values below s round to x and above s round to x'. For the successive irreducible fractions $\frac{p}{q} < \frac{r}{s}$ in K, the splitting point for the rounding Φ_K can be shown[3] to be the medQiant $\frac{p+r}{q+s}$, so Φ_K is termed medQiant rounding. Utilizing the mediant rounding Φ_K, the fixed-slash arithmetic operators which characterize closed fixed-slash approximate arithmetic can now be described.

The monadic operators absolute value, negation, and inverse are closed in the fixed-slash system characterized by K and are implemented for the word format of Figure 1 as follows.

Monadic Operator	Implementation
Absolute Value	Sign Bit ← 0
Negation	Complement Sign Bit
Inverse	Swap Numerator and Denominator N Bit Fields

Each of the dyadic algebraic operators +,-,×,÷ applied to a pair of fractions of K(N) yields a fraction exactly representable in K(2N+1) using no more than standard integer multiplication and double length addition or subtraction. Double length registers and double length integer arithmetic is sufficient to realize closed fixed-slash approximate arithmetic in K defined as follows for all (p,q), (r,s) ε K. The necessary sign bit manipulations for signed fixed-slash operands are taken as an obvious and implicit extension.

Dyadic Operators

$$\begin{aligned} \frac{p}{q} \boxed{\times} \frac{r}{s} &= \Phi_K\left(\frac{pr}{qs}\right) \\ \frac{p}{q} \boxed{\div} \frac{r}{s} &= \Phi_K\left(\frac{ps}{qr}\right) \\ \frac{p}{q} \boxed{+} \frac{r}{s} &= \Phi_K\left(\frac{ps+qr}{qs}\right) \\ \frac{p}{q} \boxed{-} \frac{r}{s} &= \Phi_K\left(\frac{ps-qr}{qs}\right) \end{aligned} \tag{5}$$

Note that an implementation of the dyadic operators $\boxed{\times}$ and $\boxed{\div}$ first require two single length integer multiplications which may be computed in parallel followed by an implementation of the rounding Φ_K applied to an argument with at most 2N bits each in the numerator and denominator. The operators $\boxed{+}$ and $\boxed{-}$ require three single length integer multiplications which may be computed in parallel, followed by a double length integer addition or subtraction, with an implementation of $\Phi_{K(N)}$ then necessary for an argument with at most 2N+1 bit numerator and 2N bit denominator.

Thus the overall complexity and feasibility of binary fixed-slash arithmetic depends on the efficiency of implementation of a special purpose "double length" to "single-length" mediant rounding Φ_K.

III. The Binary Shift-Subtract GCD and Binary Convergent Algorithms

It is evident from the Euclidian-Convergent Algorithm of Section II that the b_i, p_i and q_i may be computed in parallel. For binary representation

$$a_i = \sum_j d_{ij} 2^j \text{ where } d_{ij} = 0,1 \text{ for all } i,j,$$

so then

$$\left.\begin{aligned} b_i &= b_{i-2} - \sum_j d_{ij}\,(2^j b_{i-1}) \\ p_i &= p_{i-2} + \sum_j d_{ij}\,(2^j p_{i-1}) \\ q_i &= q_{i-2} + \sum_j d_{ij}\,(2^j p_{i-2}) \end{aligned}\right\} \quad i = 0,1,\ldots,n. \tag{6}$$

From (6) the following observations are pertinent to a binary implementation of a convergent algorithm.

(1) Regarding hardware: For each i, the bits d_{ij} may be determined (most significant bit first) from b_{i-2} and b_{i-1} by the shift and subtract procedure of a standard binary division implementation. As each bit d_{ij} is determined, p_i and q_i may then be accumulated in parallel to b_i by a comparable shift and add procedure. Thus only binary shift, add and subtract are required.

(2) Regarding storage: The a_i need not be explicitly computed, and only two successive values, b_{i-2} and b_{i-1}, need be saved in computing the b_i's. If only the K-finest convergent is desired as specified for mediant conversion, then only two successive values p_{i-2}, p_{i-1} and q_{i-2}, q_{i-1} need be saved during the recursive computation of the desired convergent.

We now incorporate these observations into two algorithms. The first, Algorithm SS, simply computes the greatest common divisor of u and v, defined by gcd(u,v) = max {w|w divides u, w divides v}. The second, Algorithm BC, computes the last representable (K-finest) convergent, thus realizing the mediant rounding Φ_K.

Algorithm SS (Binary Shift-Subtract GCD)

Given $u \geq 1$, $v \geq 0$, this algorithm determines gcd(u,v).

```
     while v > 0 do
LO:  begin k: = 0;
          while nsb(u) > nsb(v) do
     L1:  begin k: = k+1;  v: = 2×v end;
     L2:  loop d: = u-v;
               if d ≥ 0 then u: = d;
          exit if k = 0;
               v: = v/2;  k: = k-1;
          end;
          interchange (u,v);
     end
     gcd: = u;
```

Comments:

(1) The loop L1 left-shifts v until its most significant bit is in the same position as that of u, using the function nsb to encode the number of significant bits (or equivalently the position of the most significant bit). In a hardware realization this test can be realized as a one bit enable/disable.

(2) Note that the inner loop L1 is executed only one less time than the inner loop L2, so we may consider them together as the minor cycle, each cycle consisting of one subtraction and two shifts (implementing k by shifting).

Algorithm BC (Binary Convergent in K for r/s)

Given $r \geq 0$, $s \geq 1$ and the fixed-slash system K, this algorithm determines the K-finest convergent p/q to r/s, i.e. the mediant rounding $\Phi_K(\frac{r}{s})$. Parallel execution of statements is expressed using and as separator, with the semicolon denoting standard sequential execution.

```
  b':=r and p':=0 and q':=1 and k:=0 and
  b:=s and p:=1 and q:=0;
  while {max(b,b') not yet normalized} do
       begin b:=2×b and b':=2×b' end;
  while (b > 0) do
LO:begin
    while {b not yet normalized} ∧ {(p,q)εK} do
  L1:begin k:=k+1 and b:=2×b and p:=2×p and
     q:=2×q end;
  L2:loop
       d:=b'-b and p":=p'+p and q":=q'+q;
       if d ≥ 0 then
            begin b':=d and p':=p" and q'=q" end;
     exit if k=0;
       b'=2×b' and p:=p/2 and q:=q/2 and k:=k-1;
     end;
  exit if {(p',q') ∉ K}
     interchange (b,b') and interchange (p,p') and
     interchange (q,q')
  end:
  {On exit p/q is the last representable convergent}
```

Comments:

(1) Note that the left-shifting of b' during the loop L2 may move the most significant bit of b' into the "sign-position", but this does not affect the result when the algorithm is implemented in two's complement arithmetic.

(2) For $\frac{r}{s} = [a_0, a_1, \ldots, a_m]$ and $\Phi_K(\frac{r}{s}) = \frac{p_n}{q_n}$ the outer loop (LO) will be executed n+2 times (if n < m and n+1 times if n = m), the last time being the (partial) computation of $\frac{p_{n+1}}{q_{n+1}}$. In the i'th major cycle b_i, p_i and q_i are being computed in the inner loops (L1 and L2), which will be analyzed in the next sections.

The necessary circuitry for a hardware realization of Algorithm BC consists basically of one 2N+2 bit "subtractor", plus two N+1 bit adders. Figure 2 shows these components together with the necessary control for a slightly different version of the loop L2 (based on right-shifting of b instead of left-shifting of b', and shifting for k).

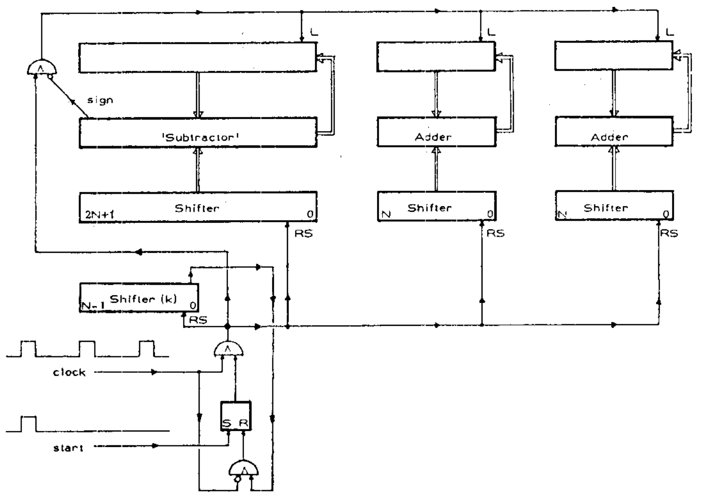

Figure 2.

It is worth noticing that the LS Algorithm by Brent[7] also might be used for a Euclidian-Convergent Algorithm by a proper extension. The LS Algorithm basically skips over zero-bits of the quotient a_i, hence executes fewer cycles than the SS or

BC Algorithms, but each cycle requires one "normalize" and two or three subtractions in comparison to one subtraction and two shifts per cycle in Algorithm SS. Following the basic idea of the LS Algorithm, the BC Algorithm might be improved by enclosing in a loop the "single-shifts" of b', p, q and k at the end of loop L2. This new loop has to execute at least once, but would continue until b' is normalized or until k = 0, thus avoiding some trial subtractions. The comparatively small expected number of subtractions avoided does not seem to justify the added complexity for the current analysis.

IV. Upper Bounds on the BC Algorithm

Returning to the recursive construction of the p_i and q_i, if for any x and y a sequence $s_0, s_1, \ldots, s_n$ is constructed from the linear recurrence

$$s_i = a_i s_{i-1} + s_{i-2}$$

with initial values $s_{-1} = y$ and $s_{-2} = x$ then

$$s_i = xq_i + yp_i.$$

This result may be utilized to characterize each b_i of the gcd algorithm in terms of p_i, q_i, p, and q. Taking x = p and y = -q we have

$$s_i = pq_i - qp_i.$$

For the standard convergent computation initial conditions $b_{-1} = q$ and $b_{-2} = p$, we have

$$b_i = -a_i b_{i-1} + b_{i-2},$$

hence

$$b_i = (-1)^i s_i = (-1)^i (pq_i - qp_i).$$

We can thus state the following lemma concerning the relation of the b_i to the (p_i, q_i).

Lemma 3: The sequence of remainders $b_0, b_1, \ldots, b_n = 0$ of the gcd algorithm applied to (p,q) satisfy the following relations to the convergents $\frac{p_i}{q_i}$ of $\frac{p}{q}$:

(i): $b_i = (-1)^i (pq_i - qp_i)$ or (ii): $\frac{p}{q} - \frac{p_i}{q_i} = (-1)^i \frac{b_i}{qq_i}$.

Furthermore for $0 \leq i \leq n-1$,

(iii) $q_{i+1} \leq \frac{q}{b_i} < q_i + q_{i+1}$,

(iv) $p_{i+1} \leq \frac{p}{b_i} \leq p_i + p_{i+1}$.

Proof of (iii): For i < n, $\frac{p_i + p_{i+1}}{q_i + q_{i+1}}$ (the mediant of $\frac{p_i}{q_i}$ and $\frac{p_{i+1}}{q_{i+1}}$) must fall between $\frac{p}{q}$ and $\frac{p_i}{q_i}$, and using Theorem 1(iv),

$$\left|\frac{p_i + p_{i+1}}{q_i + q_{i+1}} - \frac{p_i}{q_i}\right| < \left|\frac{p}{q} - \frac{p_i}{q_i}\right| \leq \left|\frac{p_{i+1}}{q_{i+1}} - \frac{p_i}{q_i}\right|,$$

then using Theorem 1(iii),

$$\frac{1}{(q_i + q_{i+1})q_i} < \left|\frac{p}{q} - \frac{p_i}{q_i}\right| \leq \frac{1}{q_{i+1} q_i},$$

which together with (ii) proves (iii). (iv) is proved analogously. ▌

To estimate the timing of Algorithm BC we will start by establishing upper bounds based on the following lemma. A minor cycle of Algorithm BC (as in Algorithm SS) will contain two shift and one subtraction time periods.

In the following log x will denote logarithms to the base 2.

Lemma 4: The accumulated number of minor cycles S_n in Algorithm BC to compute the convergent $\frac{p_n}{q_n}$ of $\frac{p}{q} = [a_0, a_1, \ldots, a_m]$, $n \leq m$, is bounded by:

$$\log p_n + n \leq S_n \leq \log p_n + n + 3 \quad \text{for } p \geq q.$$

Proof: Introducing the function (length in bits)

$$\ell(x) = \begin{cases} \lfloor \log x \rfloor + 1 & \text{for } x > 0, \\ 1 & \text{for } x = 0, \end{cases}$$

the number of minor cycles in the i'th major step for $0 \leq i \leq n$, assuming $p \geq q$, is

$$k_i = \ell(b_{i-2}) - \ell(b_{i-1}) + 1,$$

hence $$S_n = \sum_0^n k_i = \ell(p) - \ell(b_{n-1}) + (n+1).$$

Then

$$\log \frac{p}{b_{n-1}} + n \leq S_n \leq \log \frac{p}{b_{n-1}} + n + 2.$$

Using Lemma 3 (iv) the lemma is proved. ▌

The total number of minor cycles in Algorithm BC applied to (p,q) is denoted by $S(\frac{p}{q})$ and includes the number S_n to compute $\frac{p_n}{q_n}$ plus the number sufficient to indicate that $\frac{p_{n+1}}{q_{n+1}} \notin K$. Necessary bounds on n (number of major cycles) are derived utilizing the classical theory of Fibonacci numbers[4,9].

Theorem 5: For the fixed-slash system K (N numerator and N denominator bits), the number of minor cycles $S(\frac{p}{q})$ of Algorithm BC to determine the mediant rounding $\Phi_K(\frac{p}{q})$ satisfies:

(i) for any rational $\frac{p}{q} > 0$,
$S(\frac{p}{q}) \leq cN+7$,

(ii) for some rational $(p,q) \in K$,

$S(\frac{p}{q}) \geq cN-3$,

where $c = \left(1 + \frac{1}{\log\left(\frac{\sqrt{5}+1}{2}\right)}\right) = 2.4404\ldots$.

Proof: Fibonacci numbers are defined by the recurrence $f_0=0$, $f_1=1$, and $f_n=f_{n-1}+f_{n-2}$ for $n \geq 2$. Some relevant properties of Fibonacci numbers[4,9] are:

(a) $\frac{f_{n+1}}{f_{n+2}} = [a_0=0, a_1=1, a_2=1, \ldots, a_{n-1}=1, a_n=2]$

for all $n \geq 1$,

(b) the convergents of $\frac{f_{n+1}}{f_{n+2}}$ for any $n \geq 1$ are

$$\frac{p_i}{q_i} = \frac{f_i}{f_{i+1}} \text{ for } 0 \leq i \leq n-1,$$

$$\frac{p_n}{q_n} = \frac{f_{n+1}}{f_{n+2}} \text{ for } i = n,$$

(c) $f_n = \left\lfloor \frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2}\right)^n + \frac{1}{2} \right\rfloor$ for any $n \geq 0$.

Choose n such that

$$\frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2}\right)^{n+1} \leq 2^{N-1} < \frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2}\right)^{n+2},$$

and note from (c) that $(f_{n+1}, f_{n+2}) \in K$.

Thus

$$(n+2) \log\left(\frac{1+\sqrt{5}}{2}\right) - \log\sqrt{5} > N-1,$$

so

$$n > \frac{1}{\log\left(\frac{1+\sqrt{5}}{2}\right)} \cdot N-2 \quad (7)$$

Since $(f_{n+1}, f_{n+2}) \in K$,

$$S\left(\frac{f_{n+1}}{f_{n+2}}\right) = S_n\left(\frac{f_{n+1}}{f_{n+2}}\right) = 1 + S_{n-1}\left(\frac{f_{n+2}}{f_{n+1}}\right),$$

and using Lemma 4

$$S\left(\frac{f_{n+1}}{f_{n+2}}\right) \geq n + \log f_{n+2} \geq N + n - 1, \quad (8)$$

so then (7) and (8) yield (ii) of the theorem for

$$\frac{p}{q} = \frac{f_{n+1}}{f_{n+2}}.$$

To prove (i) note from (a), (b) and the requirements on the a_i in the definition (3) that the convergents to any rational $\frac{p}{q}$ must satisfy

$$p_i \geq f_i, \quad q_i \geq f_{i+1} \quad \text{for all } i \geq 0.$$

Hence

$$2^N - 1 \geq q_n \geq f_{n+1} = \left\lfloor \frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2}\right)^{n+1} + \frac{1}{2} \right\rfloor,$$

so

$$2^N > \frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2}\right)^{n+1},$$

and

$$n < \frac{1}{\log\left(\frac{1+\sqrt{5}}{2}\right)} \cdot (N + \log\sqrt{5}) - 1. \quad (9)$$

In the last major cycle of Algorithm BC as applied to an arbitrary rational $\frac{p}{q}$, note that after the normalization loop L1, irrespective of the value of a_{n+1}, the value of the variable k is such that $p_n, q_n < 2^{N-k+1}$. Then using Lemma 4 and adding one minor cycle for possible initial switching of p and q,

$$S\left(\frac{p}{q}\right) \leq S_n\left(\frac{\max\{p,q\}}{\min\{p,q\}}\right) + k + 2 \leq N + n + 6,$$

and (9) and (10) yield statement (i) of the theorem. |

Although the theorem gives an upper bound on the number of minor cycles valid for mediant rounding of any rational $\frac{p}{q}$, the timing will depend on the magnitude of p and q since a subtraction is part of the minor cycle (the timing of which increases with the logarithm of the number of bits in standard adders with carry look ahead).

V. Average Case Complexity for the BC and SS Algorithms

The average value of the number of steps in Algorithm BC will depend on the distribution of numbers to be rounded. Most existing analysis of the Euclidian algorithm and of continued fractions is based on some sort of uniform distribution. For our purposes, if we restrict our analysis to the interval zero to unity (and assume symmetry around unity) some uniform distribution might be appropriate, but this is probably not the distribution of numbers occuring in actual computations.

In discussing gcd algorithms it is usually assumed that u and v are uniformly and independently distributed, $u \leq n$ and $v \leq n$, and asymtotic results on the average value of the number of steps in computing gcd (u,v) are obtained as $n \to \infty$ (e.g. [6,7]). For the fixed-slash system, n is fixed (but large, $n=2^N-1$). The uniform distribution relevant to a fixed-slash number system could associate equal probability to all $\frac{u}{v}$, $u < 2^{2N+1}$ and $v < 2^{2N}$ (all temporary results to be rounded), or possibly a uniform distribution for real x in the interval [0,1] with $\frac{1}{x}$ uniform for $x > 1$.

In the latter case, if $x \in [0,1]$, the probability of rounding into a particular irreducible fraction $\frac{p}{q}$ is proportional to the length of the interval rounded into $\frac{p}{q}$ by mediant rounding, which may be simulated.

For the average number of steps in Algorithm BC, we proceed similarly to the average case analysis of gcd algorithms and assume that the probability of rounding into $\frac{p}{q}$ is proportional to the multiplicity $m(p,q)$ of that fraction, which is equivalent to associating the same probability $\frac{1}{n^2}$ with any point (p,q) in

$$K_n = \left\{(p,q) \mid 1 \le p \le n,\ 1 \le q \le n\right\},\ \text{where}\quad n = 2^N - 1,$$

and applying Algorithm BC to all points in K_n. Note that when restricted to $(p,q) \in K_n$, Algorithms BC and SS have the same number of major and minor cycles.

To obtain an estimate of the average number of steps in Algorithm BC (or SS) utilizing Lemma 4, we will need the average value of the number of major cycles and the average value of $\log(\max\{p,q\})$ over all $(p,q) \in K$. Knuth under the same distribution, (u,v) uniform in K_n, determines an heuristic expression equivalent to the number of major cycles in Algorithm BC (or SS),[6]

$$\frac{12(\ln 2)^2}{\pi^2}\log n + O(1). \tag{11}$$

Thus we need an expression for the average value of $\log(\max\{p,q\})$ that is

$$M = \frac{1}{n^2}\sum_{\substack{K_n \\ \frac{p}{q}\text{ irreducible}}} \log(\max\{p,q\})\cdot m(p,q),$$

or equivalently

$$M = \frac{2}{n^2}\sum_{1\le p\le q\le n} \log\left(\frac{q}{\gcd(p,q)}\right) \tag{12}$$

Evaluation of M is aided by the following lemmas.

Lemma 6: If q has the prime factoization:

$$q = q_1^{\alpha_1} q_2^{\alpha_2} \cdots q_k^{\alpha_k},$$

then

$$\prod_{p=1}^{q} \gcd(p,q) = \left[\prod_{i=1}^{k} q_i^{\left(\frac{1-q_i^{-\alpha_i}}{q_i-1}\right)}\right]^q .$$

Proof: For each i and $j = 1,2,\ldots,\alpha_i$, there are exactly q/q_i^j values of p, $1 \le p \le q$ which contain q_i to some power greater than or equal to q_i^j. Thus

$$\sum_{j=1}^{\alpha_i} \frac{q}{q_i^j} = q/q_i + q/q_i^2 + \cdots + q/q_i^{\alpha_i}$$
$$= q(1-q_i^{-\alpha_i})/(q_i-1)$$

is an exhaustive count of occurences of q_i in $\prod_{p=1}^{q}\gcd(p,q)$, hence the lemma. ∥

Lemma 7: There exists $c > 1$ such that

$$\pi_n = \prod_{p\le q\le n} \gcd(p,q) \le c^{n^2} \text{ for all } n \ge 1.$$

Proof: From the previous lemma

$$\pi_n \le \prod_{q=1}^{n}\prod_{i=1}^{k_q} q_i^{q/(q_i-1)}$$
$$= \prod_{j=1}^{m}\prod_{\ell=1}^{\lfloor n/p_j\rfloor} p_j^{\ell\cdot p_j/(p_j-1)}$$

where $p_1, p_2, \ldots, p_m$ is the list of all primes $\le n$. From $\pi_n \le \prod_{j=1}^{m} p_j^{p_j(1+2+\ldots+\lfloor n/p_j\rfloor)/(p_j-1)}$ it follows since $p_j \le n$ that

$$\pi_n \le \prod_{j=1}^{m} p_j^{n^2/p_j(p_j-1)}$$
$$\le \left[\prod_{j=1}^{\infty} p_j^{\frac{1}{p_j(p_j-1)}}\right]^{n^2} = c^{n^2}$$

since the infinite product is convergent. ∥

Returning to the expected value M we may now state from (12) that

$$M = \frac{2}{n^2}\left[\sum_{q=1}^{n} q\log q - \log \pi_n\right]$$

and estimating the sum by an integral, with Lemma 7,

$$M = \log n + O(1) \text{ for } n \to \infty .$$

Assuming the validity of the heuristically derived expression (11) we have as a corollary: For u,v chosen uniformly on $1 \le u \le n$, $1 \le v \le n$, the average number of minor cycles (i) in Algorithm SS for computing $\gcd(u,v)$, and (ii) in Algorithm BC for computing $\Phi_{K_n}(\frac{u}{v})$, are both given asymptotically by

$$\bar{S}_n = (1 + 12(\ln 2/\pi)^2)\log n + O(1)$$
$$= 1.58416\ldots\log n + O(1). \tag{13}$$

To get an estimate of the O(1) term for $\bar{S}$ in Algorithm SS, gcd(u,v) was computed for all $0 < v < u < n$ for the cases $n = 2^i$, $i=1,2,\ldots, 13$. In the following table $\sum_n$ is the accumulated number of minor cycles, $\bar{S}'_n$ is the average value,

$\Delta_n = \bar{S}'_n - \bar{S}'_{n-1}$ (which supposedly converges to the constant 1.58416...) and $c'_n = S'_n - \Delta_n \cdot \log n$ (which approximates the O(1) term).

Table 1: Minor Cycles in Algorithm SS

n	$\log n$	Σ_n	$\bar{S}'_n$	Δ_n	c'_n
2	1	4	0.800000	–	–
4	2	20	1.428571	0.628571	0.171428
8	3	110	2.500000	1.071426	-0.714285
16	4	572	3.763157	1.263157	-1.289473
32	5	2986	5.332142	1.568984	-2.512781
64	6	14820	6.912313	1.580170	-2.568710
128	7	71452	8.522423	1.610110	-2.748347
256	8	335694	10.125904	1.603481	-2.701945
512	9	1547094	11.734632	1.608727	-2.743918
1024	10	7010100	13.331647	1.597014	-2.638494
2048	11	31343096	14.923692	1.592044	-2.588802
4096	12	138613216	16.511889	1.588197	-2.546477
8192	13	607533568	18.099283	1.587393	-2.536838

The tabulated values of Δ_n give empirical evidence for the constant 1.58416... . To compensate for the restriction $v < u$ regarding the possible initial interchange of (v,u) in Algorithm SS we will have to add 0.5 to $\bar{S}'_n$ and c'_n to get the proper values of $\bar{S}_n$ and c_n, hence we may state that

$$\bar{S}_n \simeq 1.5842 \log n - 2.0.$$

During the computation the number of major cycles was also accumulated, and the equivalent table confirmed the approximation stated by Knuth[6]:

$$\bar{T}_n \simeq 0.5842 \log n + 0.06.$$

For fixed-slash arithmetic we are essentially interested in application of mediant rounding to members of K_{n^2} rounding to K_n. A simulation of the appropriate uniform distribution was performed for this case by generating pseudo random pairs (u,v) in K_{n^2} and applying Algorithm BC to round $\frac{u}{v}$ into $\frac{p}{q}$, $(p,q) \in K_n$ (50 * n random pairs (u,v) for each value of n), yielding the following table.

Table 2: Algorithm BC, Minor Cycles (simulation)

n	$\log n$	Σ_n	$\bar{S}_n$	$\bar{S}_n - \bar{S}_{n-1}$	d_n
256	8	194378	15.18578		2.51210
512	9	429779	16.78824	1.60246	2.53035
1024	10	941904	18.39656	1.60832	2.55446
2048	11	2045902	19.97951	1.58294	2.55320
4096	12	4416125	21.56311	1.58359	2.55259
8192	13	9482387	23.15035	1.58724	2.55562
16384	14	20264078	24.73642	1.58606	2.55748
32768	15	43124341	26.32100	1.58458	2.55785
65536	16	91439904	27.90524	1.58423	2.55788

Here $d_n = \bar{S}_n - 1.5842 \cdot \log n$, which implies that the number of minor cycles in Algorithm BC may be approximated by

$$\bar{S}_n \simeq 1.5842 \log n + 2.56,$$

which agrees very well with the theoretical value for roundings applied uniformly only to members of K_n.

Table 3: Algorithm BC, Major Cycles (simulation)

n	$\log n$	Σ_n	$\bar{T}_n$	$\bar{T}_n - \bar{T}_{n-1}$	e_n
256	8	86418	6.75140	–	2.07772
512	9	187908	7.34015	0.58875	2.08226
1024	10	406526	7.93996	0.59980	2.09786
2048	11	872606	8.52154	0.58158	2.09523
4096	12	1864690	9.10193	0.58338	2.09441
8192	13	3969253	9.69055	0.58562	2.09582
16384	14	8416799	10.27411	0.58395	2.09517
32768	15	17792665	10.85978	0.58536	2.09663
65536	16	37497851	11.44343	0.58365	2.09607

Here $e_n = \bar{T}_n - 0.5842 \log n$, hence the number of major cycles is approximately

$$\bar{T}_n \simeq 0.5842 \log n + 2.10.$$

VI. Conclusions

We may now summarize the results of the previous sections, together with those of Knuth and Brent as follows[6,7].

Table 4:

Algorithm	Classical gcd	LS gcd	SS gcd
Minor Cycle	1 division	1 normalization + 2 or 3 subtractions	2 shifts + 1 subtraction
Maximum Number of Minor Cycles	1.4404 log n	1.4404 log n	2.4404 log n
Average Number of Minor Cycles (with O(1) term)	0.58416 log n + 0.06	0.87581 log n - 1.40	1.58416 log n - 2.00

Algorithm SS is hence a candidate for the fastest hardware realization of mediant rounding in the form of Algorithm BC, subject as demonstrated in Section III to utilization of appropriate parallelism in the hardware.

Also a microcoded implementation might be interesting, at least as an experimental vehicle for further studies on the use of fixed- or floating-slash arithmetic in practical computations. For this purpose we have coded Algorithm BC in microcode for the MATHILDA processor for the fixed-slash representation with $N = 31$ (the wordsize of the processor is 64 bits)[8]. The major loop consists of 11 instructions, 4 of which form the minor loop (L2). The prenormalization is realized by shifting in a barrel-shifter (i.e. no loop) outside the minor loop. In the minor loop only 2 instructions are executed in the case where no add/subtract has to take place (i.e. corresponding to a zero bit in the quotient). A little computation on the frequency of execution, and the 3MHz instruction frequency of the processor, yields an average execution time of approx. 95 μS, which is certainly not very fast for a rounding algorithm.

However almost half of the time is spent in the major loop, outside the minor loop, mostly doing quite trivial data manipulation. Also parallelism was not fully exploited, as only the two 31 bit additions were done in parallel in the microcoded version.

Returning to a hardwired realization, a minor cycle time of say 50 nS can easily be achieved (standard TTL logic, e.g. the 74 series). Adding an overhead of say 15 nS for the remaining part of the major loop, the average execution time for the Algorithm BC amounts to approx. 2.7 μS, again with $N = 31$. If we furthermore construct one 32×32 bit multiplier out of four 16×16 bit multipliers (available on one chip and with a multiply time of 180 nS), the three multiplications and the additions necessary for the addition of two fixed-slash numbers can be realized within 0.8 μS, i.e. the total add time will average 3.5 μS. And this is all in TTL logic, which is not the fastest logic available today, and not utilizing all possible parallelism. An improvement by a factor of 5, or possibly even 10, does not seem impossible, by using other technologies and more integration of components.

We may then conclude that, although Algorithm BC for mediant rounding is a quite complicated algorithm, it can be realized at a speed which makes fixed-slash (and possibly also floating-slash) arithmetic viable.

Bibliography

[1] D. W. Matula: "Fixed-Slash and Floating Slash Rational Arithmetic", Proceedings of the 3rd IEEE Symposium on Computer Arithmetic, Dallas, November 1975, pp. 90-91.

[2] D. W. Matula & P. Kornerup: "Finite Precision Rational Arithmetic, Part 1: The Number Systems", in preparation.

[3] P. Kornerup & D. W. Matula: "Finite Precision Rational Arithmetic, Part II: The Arithmetic", in preparation.

[4] G. H. Hardy & E. M. Wright: "An Introduction to the Theory of Numbers", Clarendon Press, Oxford 1954.

[5] U. Kulisch: "An Axiomatic Approach to Rounded Computations", Numerische Mathematik 18 (1971), pp. 1-17.

[6] D. E. Knuth: "The Art of Computer Programming", Vol. 2/Seminumerical Algorithms, Addison-Wesley 1969.

[7] R. P. Brent: "Analysis of the Binary Euclidian Algorithm" in J. Traub(ed.) "Algorithms and Complexity", Academic Press, 1976, pp. 321-355.

[8] P. Kornerup & B. D. Shriver: "An Overview of the MATHILDA Processor", SIGMICRO Newsletter, January 1975.

[9] D. E. Knuth: "The Art of Computer Programming", Vol. 1/Fundamental Algorithms, Addison-Wesley, 1968.