Precision of Semi-Exact Redundant Continued Fraction Arithmetic for VLSI

Oskar Mencer, Martin Morf, Michael J. Flynn

Technical Report : CSL-TR-00-791

February 2000

Precision of Semi-Exact Redundant Continued Fraction Arithmetic for VLSI

by

Oskar Mencer, Martin Morf, Michael J. Flynn

Technical Report : CSL-TR-00-791

February 2000

Computer Systems Laboratory
Department of Electrical Engineering and Computer Science
Stanford University
William Gates Computer Science Building, 4A-408
Stanford, California 94305-9040
Email: pubs@shasta.stanford.edu

Abstract

Continued fractions (CFs) enable straightforward representation of elementary functions and rational approximations. We improve the positional algebraic algorithm, which computes homographic functions such as $y = \frac{ax+b}{cx+d}$, given redundant continued fractions x, y, and integers a, b, c, d. The improved algorithm for the linear fractional transformation produces exact results, given regular continued fraction input. In case the input is in redundant continued fraction form, our improved linear algorithm increases the *percentage of exact results* with 12-bit state registers from 78% to 98%. The maximal error of non-exact results is improved from $\sim\ 1$ to 2^{-8}. Indeed, by detecting a small number of cases, we can add a final correction step to improve the guaranteed accuracy of non-exact results. We refer to the fact that a few results may not be exact as "Semi-Exact" arithmetic. We detail the adjustments to the positional algebraic algorithm concerning register overflow, the virtual singularities that occur during the computation, and the errors due to non-regular, redundant CF inputs.

Key Words and Phrases: regular continued fractions, computer arithmetic, rational arithmetic

Contents

List of Figures

1 Motivation

We start by defining some continued fraction(CF) forms. *Finite continued fractions* are rational numbers that are constructed as follows: for $a_i, b_i \in \mathcal{R}$

$$\frac{A_n}{B_n} = a_0 + \frac{b_1}{a_1 + \frac{b_2}{a_2 + \cdots \frac{b_n}{a_n}}} = a_0 + \frac{b_1|}{|a_1} + \frac{b_2|}{|a_2} + \cdots + \frac{b_n}{a_n}$$

Simple continued fractions form a special case with partial quotients $b_i = 1$. We use the series notation $[a_0; a_1, \ldots a_n]$ to denote *simple continued fractions.*

Regular continued fractions are simple continued fractions with all $a_i \in \mathcal{N}^+$, except $a_0 \in \mathcal{N}$. *Redundant continued fractions*[18] use a finite representation of partial quotients and therefore include zero, i.e. $-2^N \leq a_i \leq 2^N - 1$.

Redundant continued fractions connect uniformly distributed, binary numbers to continued fractions. CFs are at the basis of rational approximation theory. The representation of rational numbers by CFs connects number representation to transcendental functions. Although we are still missing a general theory explaining the connection between transcendental functions and continued fractions([8], see Introduction by Peter Henrici), we can already take advantage of the known, isolated "gems" of continued fraction expansions such as:

$$\tan(x) = \left[0; \frac{1}{x}, -\frac{3}{x}, \frac{5}{x}, -\frac{7}{x}, \frac{9}{x}, -\frac{11}{x}, \ldots\right] \tag{1}$$

In this paper we explore arithmetic within a continued fraction representation of rational numbers. More specifically, we improve the precision of known algorithms to compute $T_1 = \frac{ax+b}{cx+d}$, and investigate higher degree polynomials and conversion of simple continued fractions to redundant continued fractions.

Ultimatively, we envision continued fractions as a candidate for the computer engineers "bag of tricks", such as the logarithmic number system, the Fast-Fourier Transform, run-length encoding, etc.

2 Continued Fraction Theory

This section provides basic continued fraction theory for the reader not familiar with continued fractions.

A finite continued fraction with i partial quotients can always be transformed into a ratio $\frac{A_i}{B_i}$ with:

$$A_i = a_i A_{i-1} + b_i A_{i-2} \tag{2}$$
$$B_i = a_i B_{i-1} + b_i B_{i-2} \tag{3}$$

where $\frac{A_{i-1}}{B_{i-1}}$ corresponds to the value of the same continued fraction without the i^{th} partial quotient. Initial conditions are $A_0 = a_0$, $B_0 = 1$, $A_{-1} = 1$, and $B_{-1} = 0$.

Equations 2,3 can also be expressed as:

$$\begin{pmatrix} A_i \\ B_i \end{pmatrix} = \begin{pmatrix} A_{i-1} & A_{i-2} \\ B_{i-1} & B_{i-2} \end{pmatrix} \begin{pmatrix} a_i \\ b_i \end{pmatrix} \tag{4}$$

which relates the representation of CFs to the algorithms investigated in this paper.

The following equivalence shows how to convert general CFs to simple CFs. This algorithm will be useful for applying known results of general CFs to simple CFs.

Equivalence 1 *(Perron [6]) From general continued fractions to simple continued fractions:*

$$a_0 + \frac{b_1|}{|a_1} + \frac{b_2|}{|a_2} + \frac{b_3|}{|a_3} + \cdots \equiv$$
$$a_0 + \frac{1\ |}{|\frac{a_1}{b_1}} + \frac{1\ |}{|\frac{b_1}{b_2}a_2} + \frac{1\ |}{|\frac{b_2}{b_1 b_3}a_3} + \frac{1\ |}{|\frac{b_1 b_3}{b_2 b_4}a_4} + \cdots$$

Converting simple continued fractions with positive and negative integer quotients to regular continued fractions is achieved by applying the following equivalence.

Equivalence 2 *(see [3][6][17]) For any fragment of a simple continued fraction with positive and negative partial quotients,*
$[\ldots, a_i, 1, a_{i+2}, \ldots] \equiv [a_i + 1, -a_{i+2} - 1, -a_{i+3}, -a_{i+4}, \ldots]$
$[\ldots, a_i, 0, a_{i+2}, \ldots] \equiv [a_i + a_{i+2}, \ldots]$

The theorem of Lochs gives information relating decimal numbers to regular CFs.

Theorem 3 *(Lochs [15]) For almost all irrational numbers x and their approximation in the form of a regular continued fraction, we have*

$$\lim_{n\to\infty} \frac{k_n(x)}{n} \simeq 0.9702.$$

with $k_n(x)$, the number of partial quotients, and n, the number of approximated decimal digits.

In other words, the precision of a regular continued fraction with n partial quotients corresponds roughly to the precision of a decimal number with n decimal digits. We conclude that the information content of one partial quotient is on average constant and comparable to one decimal digit.

Khinchin[5] shows the distribution of the values of partial quotients x of regular continued fractions, found by Kuzmin in 1928 (see also Knuth[9]).

$$\mathtt{Pr(x = a)} = \log_2 \left(\frac{\mathtt{(x+1)^2}}{\mathtt{(x+1)^2 - 1}} \right) \tag{5}$$

Approximation of functions with their CF expansion is based on the equivalence of series and CFs.

Equivalence 4 *Equivalence of series and continued fractions: for* $c_i \neq 0$*,(Euler[2])*

$$c_0 + c_1 + c_2 + \cdots \equiv c_0 + \frac{c_1|}{|1} - \frac{\frac{c_2}{c_1}|}{|1+\frac{c_2}{c_1}} - \frac{\frac{c_3}{c_2}|}{|1+\frac{c_3}{c_2}} - \cdots$$

One partial quotient of the CF corresponds to one term of the series; thus the precision of an n element CF is equivalent to the precision of a finite series of length n. The connection between rational approximations and simple CFs is expressed in the next equivalence.

Equivalence 5 *Given the ratio of two polynomials* $\frac{f_0(x)}{f_1(x)}$*,(Wall[7])*

$$\begin{aligned}\frac{f_0}{f_1} &= \frac{a_{00}x^n + a_{01}x^{n-1} + \cdots + a_{0n}}{a_{11}x^{n-1} + a_{12}x^{n-2} + \cdots + a_{1n}} \equiv \\ &\equiv [r_1 x + s_1, r_2 x + s_2, \cdots, r_n x + s_n]\end{aligned}$$

with all $a_{ij} \neq 0$*, and* $r_i \neq 0$*.*

Software packages such as MapleV[21] compute *minimax* coefficients a_{ij} automatically. Many elementary functions have straightforward simple continued fraction expansions – the rational equivalent to Taylor series expansion around a point x_0. For example: e^x, $\log(1+x), (1+x)^n$, and $\tan^{-1}(x)$ are shown in [17].

The following theorem on the convergence of regular continued fractions ensures the convergence of the rational approximation algorithms with regular CF input.

Theorem 6 *Every regular continued fraction converges to a real number.*

For simple continued fractions it is much harder to guarantee convergence. Using Equivalence 2 we transform simple continued fractions to positive simple continued fractions. For positive simple continued fractions we then apply the following theorem.

Theorem 7 *The value of the simple continued fraction* $[a_0; a_1, a_2, a_3, \ldots]$ *converges if all the quotients* a_i *are positive and the serie* $\sum a_i$ *diverges[6].*

Rational approximations $\frac{P(x)}{Q(x)}$ have been shown to be efficient for the approximation of elementary functions. For certain functions, rational approximations can converge faster than polynomial approximations[16], or for example CORDIC-like algorithms[22], and possibly provide a larger region of convergence. Muller[20] suggests that functions that are "highly non-polynomial" are better approximated by rational approximations. Jones and Thron ([8], p. 202) claim that eight decimal digits of arctan(1) are approximated by a continued fraction six orders of magnitude faster than by a polynomial series.

Notes on the history and sources of continued fraction theory and the following continued fraction arithmetic algorithms can be found in Appendix A.

3 Continued Fraction Arithmetic Algorithms

Before looking at implementation details, we summarize the state-of-the-art in continued fraction arithmetic algorithms. We limit our summary to material that leads to the improvements in section 4.1.

Every number representation has its natural set of operations. Residue numbers favor addition, while logarithmic numbers favor multiplication. Continued fractions favor $f = 1/x$, which is computed by $[0; a_0, a_1, \ldots]$ if $a_0 \neq 0$, and $[a_1; a_2, \ldots]$ if $a_0 = 0$. Negation is achieved by negating all partial quotients. A simple continued fraction multiplied by a constant c becomes:

$$[0; \frac{a_0}{c}, c \cdot a_1, \frac{a_2}{c}, c \cdot a_3, \frac{a_4}{c}, c \cdot a_5, \ldots] \tag{6}$$

We focus on efficient and regular computation of rational functions of redundant continued fractions with finite registers and a small quotient representation.

In general, rational functions are of the form

$$f(\vec{x}) = \frac{P(\vec{x})}{Q(\vec{x})} \tag{7}$$

where $P(\vec{x})$ and $Q(\vec{x})$ are specific $|\vec{x}|$-dimensional polynomials of the same degree. The function $f(\vec{x})$ maps $|\vec{x}|$ sets of partial quotients of simple continued fractions to partial output quotients of a simple continued fraction. The coefficients of the polynomials P, Q correspond to the state of the continued fraction arithmetic (CFA) unit. We start by examining the most simple case $y = \frac{y_1}{y_2} = T_1 = \frac{ax+b}{cx+d}$, the linear fractional transformations corresponding to the linear transformation:

$$\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x \\ 1 \end{pmatrix} \tag{8}$$

Note the similarity between the computed function, and the CF number representation in matrix form in equation 4. Hurwitz[4] already computed the homographic transformation $y = \frac{ax+b}{cx+d}$ with continued fractions x, y, for variables a to $d \in \mathcal{R}$ (see also Perron[6], Book I, Ch.4).

3.1 Algebraic Algorithms

Gosper[11] proposes the general idea behind the algebraic algorithm. As in Raney's algorithm $|\underline{A}|$ = constant for all iterations.

We compute $[o_0, o_1, o_2 \ldots] = f([x_0, x_1, x_2, \ldots])$
where x_is are the partial quotients of the simple input CF, and $o_i = f(x_i, \texttt{state})$ are the partial quotients of the output.

The algorithm requires one *state register* for each coefficient.

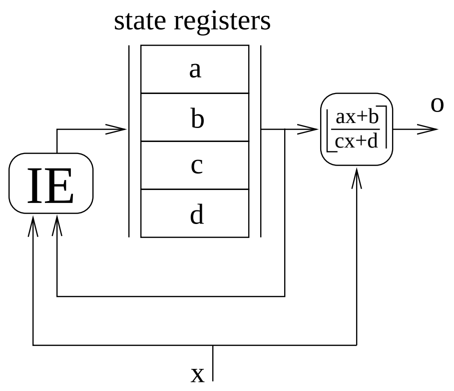

Figure 1: The figure shows the design of an arithmetic unit implementing the linear fractional transformation $T_1 = \frac{ax+b}{cx+d}$ for continued fractions. `IE` are the iteration equations shown below.

- to *consume* an input quotient x_i
  apply $T'(x) = T(x_i + \frac{1}{x})$.
- to *produce* an output quotient o_i
  apply $T''(x) = \frac{1}{T(x)-o_i}$.

Both transformations preserve the form of the homographic transformation. We can independently consume an input quotient, or produce an output quotient at each iteration. However, ensuring that quotients are consumed and produced optimally increases the overall computation time by an "order of magnitude"[17].

Vuillemin's *positional algebraic algorithm* consumes one input and produces one output at each iteration, making the computation more regular. Iteration equations and the initially proposed choice of output digits o_i are shown after Raney's algorithm.

3.2 Raney's Algorithm

Although less known, Raney[12] also computes $T_1 = \frac{ax+b}{cx+d}$, with $\underline{A} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$, for regular CF input. Raney's algorithm differs from the algebraic algorithm in the way the output digits are obtained.

First, Raney's algorithm converts $\underline{A}$ to a product of powers of matrices

$$\underline{L} = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \text{ and } \underline{R} = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}.$$

In a second pass the product of L, R matrices is converted to a regular CF. Although due to the two-pass structure, Raney's algorithm is not efficient as a computer arithmetic algorithm, we can infer that there is no natural 1-to-1 correspondence between input and output digits. Raney's algorithm explains why the positional algebraic algorithm leads to non-exact results in some cases.

As before, $|\underline{A}| =$ constant during the entire algorithm, leading to an upper limit on the values of state registers.

4 Implementation

Linear Fractional Transformation

Figure 1 shows a block diagram for a one-dimensional, linear, positional algebraic arithmetic unit. In the linear case we compute the transformation $T_1 = \frac{ax+b}{cx+d}$. Given an input quotient x_i, we choose[1] the output at each iteration $o_i = \left\lfloor \frac{ax_i+b}{cx_i+d} \right\rceil$. The iteration equations (IE) are:

$$\begin{array}{ll} \mathtt{a}' = \mathtt{cx_i} + \mathtt{d} & \mathtt{b}' = \mathtt{c} \\ \mathtt{c}' = \mathtt{ax_i} + \mathtt{b} - \mathtt{o_i}(\mathtt{cx_i} + \mathtt{d}) & \mathtt{d}' = \mathtt{a} - \mathtt{o_i c} \end{array} \tag{9}$$

Quadratic Transformations

In the quadratic case we compute the transformation $T_2 = \frac{ax^2+bx+c}{dx^2+ex+f}$, with x as the current input and o as the current output (indices i are omitted for simplicity). We first choose $o = \left\lfloor \frac{ax^2+bx+c}{dx^2+ex+f} \right\rceil$, and afterwards update the state registers as follows:

$$\begin{array}{ll} \mathtt{a}' = \mathtt{dx}^2 + \mathtt{ex} + \mathtt{f} & \mathtt{d}' = \mathtt{ax}^2 + \mathtt{bx} + \mathtt{c} - \mathtt{oa}' \\ \mathtt{b}' = 2\mathtt{dx} + \mathtt{e} & \mathtt{e}' = \mathtt{b} + 2\mathtt{ax} - \mathtt{ob}' \\ \mathtt{c}' = \mathtt{d} & \mathtt{f}' = \mathtt{a} - \mathtt{oc}' \end{array} \tag{10}$$

We can look at T_2 also as a special case of an affine transformation of the form:

$$\begin{pmatrix} y_1 \\ y_2 \\ 1 \end{pmatrix} = \begin{pmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x^2 \\ x \\ 1 \end{pmatrix} \tag{11}$$

Note that the quadratic case can be split into $T_2 = \frac{ax+b}{cx+d} \cdot \frac{ex+f}{gx+h}$, which is also possible in the next case.

In the quadratic case with two input variables, we compute the transformation $T_3(x, y) = \frac{axy+bx+cy+d}{exy+fx+gy+h}$ where x,y are inputs to the following iteration equations for the state registers. For input digits x,y, and corresponding output digit $o = \left\lfloor \frac{axy+bx+cy+d}{exy+fx+gy+h} \right\rceil$ we obtain:

$$\begin{array}{ll} \mathtt{a}' = \mathtt{exy} + \mathtt{fx} + \mathtt{gy} + \mathtt{h} & \mathtt{e}' = \mathtt{axy} + \mathtt{bx} + \mathtt{cy} + \mathtt{d} - \mathtt{oa}' \\ \mathtt{b}' = \mathtt{ex} + \mathtt{g} & \mathtt{f}' = \mathtt{ax} + \mathtt{c} - \mathtt{ob}' \\ \mathtt{c}' = \mathtt{ey} + \mathtt{f} & \mathtt{g}' = \mathtt{ay} + \mathtt{b} - \mathtt{oc}' \\ \mathtt{d}' = \mathtt{e} & \mathtt{h}' = \mathtt{a} - \mathtt{od}' \end{array} \tag{12}$$

[1] "Choose" refers to the fact that we can choose any output digit. The iteration equations adapt the state according to the chosen output quotient.

VLSI Implementation

We are interested in a hardware implementation of rational arithmetic based on continued fractions transformations such as T_1, T_2, T_3.

Partial Quotients are the digits of a number representation based on simple CFs. Vuillemin[17] investigates issues regarding computability and number representation for partial quotients of continued fractions. Trivedi[13] and Kornerup[18] suggest the use of a small set of values to represent partial quotients of continued fractions. Specifically, Kornerup suggests limiting the values of partial quotients to $\{-2, -\frac{1}{2}, 0, \frac{1}{2}, 2\}$. This reduces the computational complexity of a hardware implementation to shift-and-add. The drawback is that we can not easily guarantee convergence for redundant continued fractions (see Theorem 6). In addition, such a representation might be very wasteful for large quotients, e.g.

$\frac{1}{15} = [0; 1, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2]$

Theorem 3 leads us to consider a *4+1-bit signed digit representation for partial quotients.* We know the distribution of the partial quotients of regular CFs from the previous section. By choosing 4-bit for the magnitude of integers to represent one partial quotient we cover over 90% of the partial quotients of regular CFs. Including zero in the quotient digit set results in redundant CFs and allows us to handle quotient digit overflow by using Equivalence 2. Given a *finite* 5-bit representation of quotient digits and *finite-size* state registers, we observe the following three main *sources of error*:

1. **Overflow of State Registers**: Representing the state registers with a finite number of bits leads to frequent overflow and limits the precision of the final result even with the use of floating-point-like arithmetic. Previous work[18] assumes infinite size registers[2].

2. **Virtual Singularities**: Even if there is no real division by zero, an intermediate input quotient may cause $cx + d = 0$ (in the linear case). Aborting the algorithm in case of division by zero introduces error because the following part of the input is ignored.

3. **Overflow of Quotient Digits**:The overflow of partial quotients creates Redundant Continued Fractions[3]. For 5-bit quotient digits, any quotient digit x with $|x| > 2^4$ results in a zero digit according to Equivalence 2. Thus, the CFs are not regular any more, and the algorithm may not converge to the right value. The following conditions also cause the algorithm to produce an error:

   - A CF ending with 1.
   - strings of 1s, e.g. $[\ldots, 1, 1, 1, 1, \ldots]$.
   - An end of the form $[\ldots, x, 0, x, 0, x, 0]$ or $[\ldots, x, 0, x, 0, x]$.

[2]The paper[18] mentions that the results are true "given sufficient register lengths".

[3]An explanation of the distribution of regular continued fractions can be found in an article by Hall[10]. Hall proves that any rational number can be expressed as the sum or the product of two regular continued fractions with limited partial quotients, i.e. redundant continued fractions without zero quotients.

Given infinite resources and a converging, regular input CF, the positional algebraic algorithm converges to the exact result. However, the state registers either (1) quickly converge to the simplest form $\begin{pmatrix} |A| & 0 \\ 0 & 1 \end{pmatrix}$ for T_1, or (2) diverge very rapidly towards infinity. In case (1) we obtain an exact result. Case (2) causes an overflow of the state registers and results in an approximate result.

Although the positional algebraic algorithm produces almost always exact results if all input quotients are regular CFs, the distribution of these numbers is different from the uniform distribution of binary numbers. The rational input values to our simulations are binary numbers, converted to redundant continued fractions with quotients in $[-2^4 \ldots 2^4]$.

4.1 Improved Positional Algebraic Algorithm

We propose the following *improvements* for the linear case,T_1, based on the three sources of error, explained above:

1. **Overflow of State Registers**: The *simple* solution is to shift the coefficients a to d to the right, and continue computation to a possibly non-exact result. Raney's results (section 3.2, [12]) suggest that we might not want to create exactly one output for each input. Raney's observations lead to a better solution to state register overflow: produce an output without consuming an input. The output can be chosen to decrease the values of the state registers without introducing error.

2. **Virtual Singularities**: We choose the best possible value: $sign(ax+b) \cdot 2^4$.

3. **Overflow of Quotient Digits Creating Redundant Continued Fractions**: Redundant continued fractions allow us to pick from a variety of different continued fractions for the input to the transformation to avoid the cases that lead to non-exact results. We use equivalences such as 2.

4. **Speedup** of the convergence of the transformation matrix $\underline{A}$ to $\begin{pmatrix} |A| & 0 \\ 0 & 1 \end{pmatrix}$:
   In case $\left\lfloor \frac{a-1}{c} \right\rceil = \frac{a-1}{c}$, we choose $o = \frac{a-1}{c}$, resulting in $d = 1$.

While the last feature does not address a particular source of error, it forces $\underline{A}$ faster to its simplest form where $c = 0, d = 1$. Examining equations 9 in more detail, we see that by choosing $o \sim \frac{ax+b}{cx+d}$, $c \to 0$. Intuitively, we also want to force $d \to 1$, in order to simplify $\underline{A}$.

For $|\underline{A}| = \pm 1, \underline{A}$ converges to the identity matrix $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$. As soon as the identity matrix is reached, the tail of the input fraction is equal to the tail of the output fraction[6]; $\underline{A}$ does not change anymore, and the calculation can be terminated.

Below we show the improved **S**emi-**E**xact **P**ositional **A**lgorithm (`SEPA`) based on the positional algebraic algorithm for continued fraction arithmetic with finite state registers and redundant CFs.

SEPA Algorithm with Improvements:

```
 1. S[0]:={a,b,c,d}   //Init State Regs
 2. loop i,o from 0 to MAXLEN
 3.   if ((c*x[i]+d)==0.0)
 4.      Output[o]=sign(a*x[i]+b)*2^MAXVAL
 5.   else
 6.     if (c<>0)
 7.       if(frac((a-1)/(c))==0.0) and
 8.         (frac((a*x[i]+b)/(c*x[i]+d))<>0.0)
 9.           Output[o]=round((a-1)/c)
10.     else
11.       Output[o]=round((a*x[i]+b)/(c*x[i]+d))
12.   Compute Next State (S[o+1])
13.   if {state regs overflow}
14.     if (c<>0)
15.       Output[o]=round(a/c)
16.     else
17.       Output[o]=sign(a)*2^MAXVAL
18.   else
19.     i++; // consume input
20.   o++;   // produce output
21. endloop
22. return(Output)
```

The algorithm is shown for transformation T_1. `S[o]` stands for the state registers. Truncation of the output quotients to representable values is not shown for simplicity. `MAXLEN` is the maximal length of the input CF. `MAXVAL` is the maximal value of a redundant CF quotient – chosen in case of a *virtual singularity*. Virtual singularities occur due to the truncation of CF quotients to integers.

In the *simple* case, lines `14-17` are replaced by a right-shift(division by a power of 2) of all state-registers.

4.2 Experimental Results

We use MapleV[21] to improve continued fraction arithmetic algorithms, and simulate various implementations. Exact arithmetic enables us to study the behavior of continued fraction arithmetic algorithms with arbitrary precision, limited only by computation time.

Running the simulations of the positional algebraic algorithm gives us more insight into its behavior and accuracy over a large set of inputs. We compare the *simple* algorithm to the improved `SEPA` algorithm described above. The simple version basically follows the standard algorithm, with a floating-point-like right-shift of all state registers on register overflow.

We classify individual results into **exact** and **non-exact** results. **Exact** results are results that match the result computed with Maple's exact arithmetic. **Non-exact** results differ from Maple's exact result by some error. We present the maximal error occuring within the non-exact results, and the average error, also within the non-exact results. Section

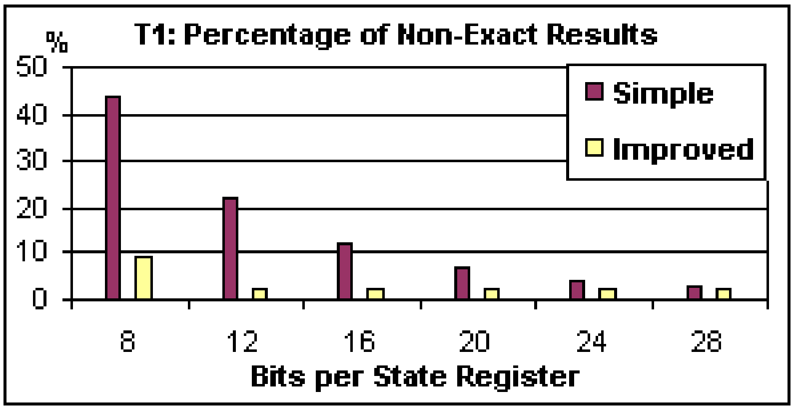

Figure 2: **Example 1:**$T_1 = \frac{ax+b}{cx+d}$; Exact results match the input CF evaluated with T_1 using Maple's exact arithmetic. The "simple" results are obtained with "shift on overflow" of state registers.

5 deals with improving the maximal error with a correction term. The improvements suggested above are primarily chosen to minimize the percentage of non-exact results.

Example 1 *Linear fractional transformation* T_1*:*
$\underline{A} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$ *with initial* $a, b, c, d \in [1 \dots 15]$*. Results are shown in figure 2.*

The accuracy of the improved results does not depend much on state register size. For reasonably sized state registers 98.5% (1.5%) of results are (non-)exact. In addition, within the 1.5% of inputs that yield non-exact results, the *average* error is about 2^{-21}, and *maximal* error is 2^{-8}. The histogram of the distribution of error within non-exact results is shown in figure 5.

Example 2 *Quadratic fractional transformation* T_2*:*
Figure 3 shows cases:

- *square function* $\rightsquigarrow T_2 = \frac{x^2}{1}$.
- *chosen case* $\rightsquigarrow T_2 = \frac{x^2+x+1}{3x^2+2x+1}$.

Quadratic transformations create quadratic growth of state register values, resulting in a stronger dependence of precision on the size of the state registers.

As a consequence, overflow of state registers occurs more often, and the improvements that worked well in the linear case (T_1) fail to improve the performance in the quadratic case (results are shown without "improvements"). Still, with 28-bit registers, $\sim 75\%$ of inputs yield exact results. Average error of the non-exact computations is about $2^{-10} - 2^{-20}$ for the square function, and $2^{-20} - 2^{-25}$ for the chosen case $(1, 1, 1, 3, 2, 1)$, depending on state register size. In both cases maximal error is ~ 1 for state register sizes larger or equal to 12 bits.

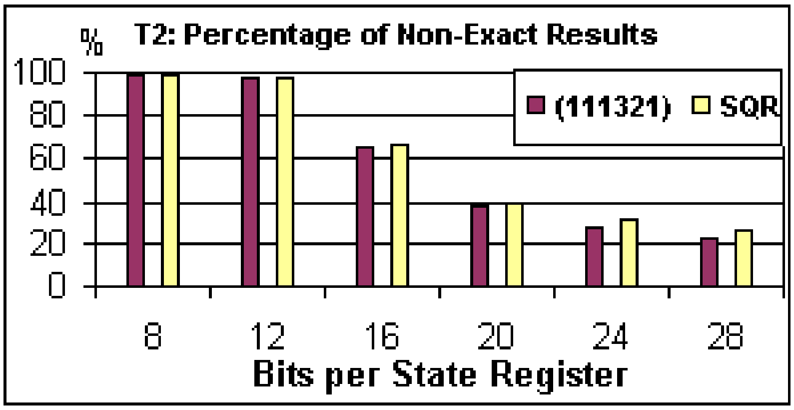

Figure 3: **Example 2:**$T_2 = \frac{x^2}{1}$, $T_2 = \frac{x^2+x+1}{3x^2+2x+1}$; Exact results match the input CF evaluated with T_2 using Maple's exact arithmetic.

Example 3 *Quadratic fractional transformation of two input variables, T_3:*

- *Multiplication* $\rightsquigarrow T_3 = \frac{xy}{1}$.
- *Addition* $\rightsquigarrow T_3 = \frac{x+y}{1}$.

Results are shown in figure 4.

As in the quadratic case with one input variable, the "improvements" of the linear case do not apply for the quadratic case T_3. Even with 28 bit registers, only about 70-80% of the results are exact. However, for the two cases shown in figure 4 the average error of the non-exact results is about 2^{-20} for register sizes larger or equal to 16 bits.

4.3 Simple Continued Fraction Inputs

Simple continued fractions consists of partial quotients $\in \mathcal{R}$. We use the identity transformation to convert simple continued fractions with rational partial quotients to redundant continued fractions – implicitly evaluating a continued fraction expansion, in our case $\tan(x)$.

Example 4 *We evaluate* $\tan(x)$ *(from equation 1) with the identity transformation* $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$, $T_1 = \frac{x}{1}$.

We observe a dependence of the accuracy of the final result on the accuracy of the input quotient. In fact, simulations show that the *average* error of the result is close to the precision of the state registers. *Maximal* error is roughly the square-root of the average error (i.e. half the bits).

Note that for simple continued fraction input the algorithm does not produce any exact results. Accuracy is now not limited by state register overflow, as much as by the loss of accuracy from truncation of fractional digits.

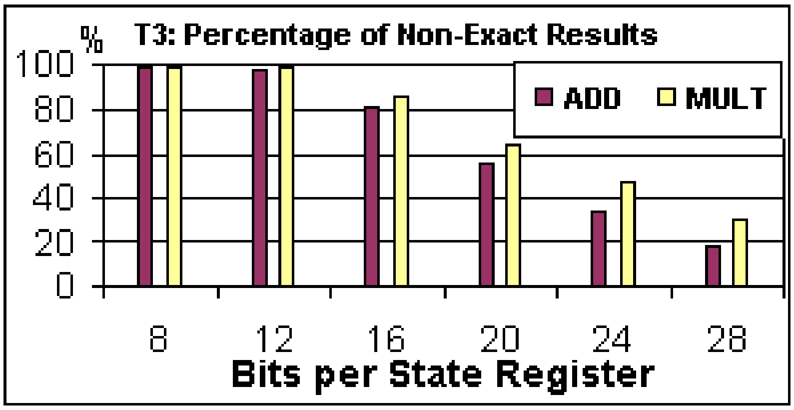

Figure 4: **Example 3:**$T_3 = \frac{xy}{1}$, and $T_3 = \frac{x+y}{1}$. Exact results match the input CF evaluated with T_3 using Maple's exact arithmetic.

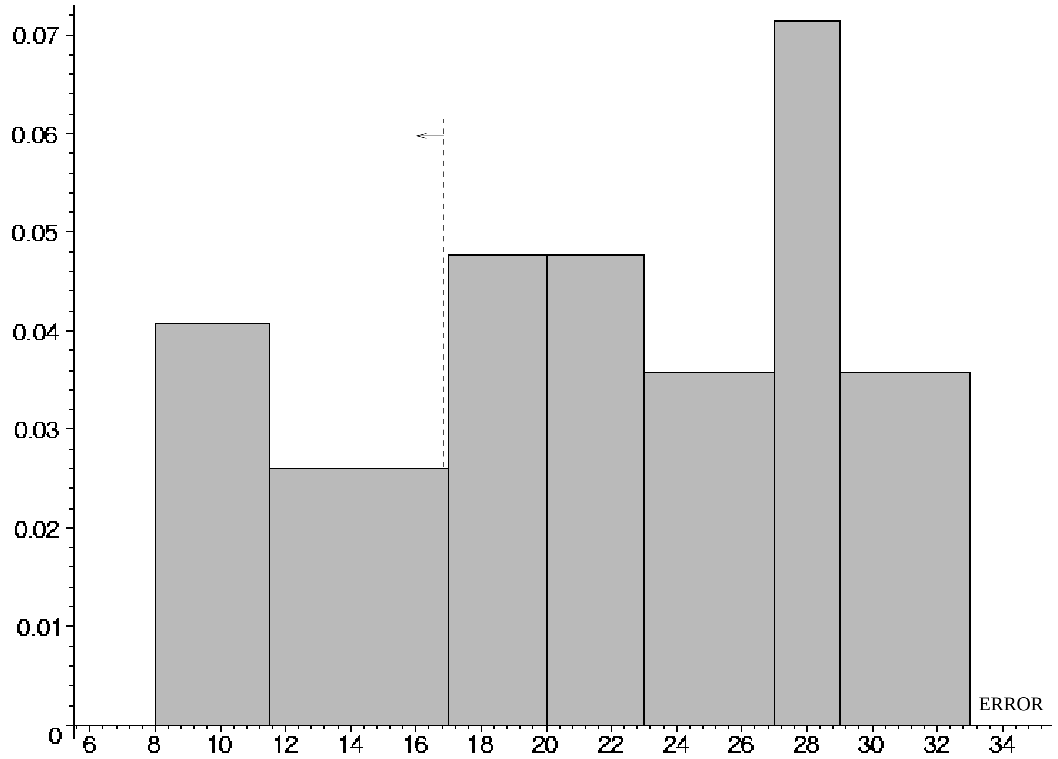

Figure 5: The figure shows the histogram of error of non-exact results (logarithmic x-axis) for $T_1 = \frac{ax+b}{cx+d}$ (example 1). Boxes contain equal amounts of data points.

It appears reasonable to expect that a converging simple continued fraction at the input would improve the accuracy (convergence) of the output. However, the following convergent, positive, simple continued fraction expansion of $\tan(x)$, obtained from equation 1, fails to improve the precision of the final result.

$$\tan(x) = [0; \frac{1}{x} - 1, 1, \frac{3}{x} - 2, 1, \frac{5}{x} - 2, 1, \frac{7}{x} - 2, 1, \ldots] \tag{13}$$

As in the case of continued fractions with integer quotients, we find no simple dependence between the convergence of the input, and the exactness of the output.

5 Final Optimization

Within the non-exact results, the proposed algorithm has a very low *average* error, but a relatively high *maximal* error. We discuss final optimization of example 1 from above.

Figure 5 shows the histogram of the distribution of the error within the 1.5% of non-exact results in the case of T_1 with 12-bit state registers, as shown in Example 1 above. We see that the error is almost uniformly distributed. In order to guarantee 16 bits of precision, we have to find a correction value for about 300 non-exact input values (out of 64K possible 16-bit inputs). A small programmable array such as a table, PLA, etc., indexed with a subset of input bits holds the 300 correction values. In case of a non-exact result, the corresponding correction value is added to the final result.

6 Conclusions

In general, finite resources limit the achievable precision of continued fraction arithmetic. The proposed improvements make it feasible to obtain exact results in 98.5% of cases for the linear fractional transformation (T_1), even with relatively small registers – making the algorithm interesting for implementation in hardware. Quadratic transformations create quadratic growth of state register values, resulting in a stronger dependence on the size of the state registers. A large percentage of non-exact results make it unrealistic to guarantee a specific precision with reasonable resouces.

The examples analyzed in this paper form the beginning of a bit-level understanding of *algebraic algorithms* for rational arithmetic. The major remaining issues before the proposed algorithms would become practical in the framework of binary arithmetic (in order of significance) are:

- **Conversion** of continued fractions to and from binary numbers limits the performance and applicability of current continued fraction arithmetic.
- The **Redundant Continued Fraction** representation limits the achievable precision by limiting the maximal range of partial quotients. Extending the maximal value of a quotient with "0" quotients leads to unacceptable growth of the number of partial quotients.

- **Quadratic growth** of state register values for quadratic transformations limits the predictability and precision of quadratic arithmetic units.

7 Acknowledgments

We thank D.W. Bump, J.T. Gill, M. Ercegovac, and K.W. Rudd for helpful comments and suggestions. This research is supported by DARPA Grant No. DABT63-96-C-0106, and Compaq Systems Research Center, Palo Alto.

Appendix A:Historical Notes

Cataldi[1] is first to mention continued fractions in 1613. The theory of continued fractions has its roots in the works of Euler[2], Lagrange[3], Legendre, Lambert, etc.

Khinchin[5] studies regular continued fractions as a number representation, and gives the distribution of values of partial quotients found by Kuzmin in 1928. Wall[7] summarizes the analytic theory of continued fractions with an emphasis on convergence and function theory. The theory of continued fractions is best organized by Perron[6].

Homographic function evaluation with simple continued fractions, as discussed in this paper, is first studied by Hurwitz ([4], 1896). Hall[10] follows up on Hurwitz's work, and Raney[12] shows a simplified algorithm based on linear algebra and finite state machines.

Meanwhile Knuth[9] considers continued fractions for semi-numerical algorithms. Gosper[11] notes how to evaluate homographic functions to CFs, given CF inputs. Vuillemin[17] formalizes and extends Gosper's work to the *algebraic* and *positional algebraic* algorithms. Kornerup[18] investigates a hardware implementation of Gosper's algorithm, and Potts[19] extends the notion of representing exact real numbers using expression trees with tensors as vertices.

References

[1] P.A. Cataldi, *Trattato del modo brevissimo di trovare la radice quadra delli numeri, et regole di approssimarsi di continuo al vero nelle radici dei numeri non quadrati, con le cause et inventioni loro,* Bologna, (Italy), 1613, see [6].

[2] L. Euler, *Introductio in analysin infinitorum I,* 1748, translated into English, Springer Verlag, New York, 1980.

[3] J.L. Lagrange, *Additions aux éléments d'algebre d'Euler,* 1798.

[4] A. Hurwitz, *Über die Kettenbrüche, deren Teilnenner arithmetische Reihen bilden,* Vierteljahrsschrift d. naturforsch. Gesellschaft, Zürich, Jahrgang 41, 1896.

[5] A. Khinchin, *Continued Fractions,* 1935, translated from Russian, The University of Chicago Press, 1964.

[6] O. Perron, *Die Lehre von den Kettenbrüchen,* Band I,II , Teubner Verlag, Stuttgart, 1957.

[7] H.S. Wall, *Analytic Theory of Continued Fractions,* Chelsea Publishing Company, Bronx, N.Y., 1948.

[8] W.B. Jones, W.J. Thron, *Continued Fractions: Analytic Theory and Applications,* Encyclopedia of Mathematics and its Applications, Vol. 11, Addison-Wesley, Reading, Mass., 1980.

[9] D.E. Knuth, *The Art of Computer Programming, Vol 2, Seminumerical Algorithms,* Addison-Wesley, Reading, Mass., 1969.

[10] M. Hall, *On the sum and product of continued fractions,* Ann. of Math. 48, 966-993, 1947.

[11] R.W. Gosper, R. Schroeppel, M. Beeler, *HAKMEM,* Continued Fraction Arithmetic, MIT AI Memo 239, Feb. 1972.

[12] G.N. Raney, *On Continued Fractions and Finite Automata,* Math. Ann. 206, 265-283, 1973.

[13] J.E. Robertson, K.S. Trivedi, *The Status of Investigations into Computer Hardware Design Based on the Use of Continued Fractions,* IEEE Trans. on Computers, Vol. C-22, No. 6, June 1973.

[14] B.C. Berndt, *Ramanujan's Notebooks,* Springer-Verlag, 1998.

[15] C. Faivre, *On decimal and continued fraction expansion of a real number,* Acta Arithmetica LXXXII.2(1997).

[16] C.T. Fike, *Computer evaluation of mathematical functions,* Englewood Cliffs, N.J., Prentice-Hall, 1968.

[17] J.E. Vuillemin, *Exact Real Computer Arithmetic with Continued Fractions,* IEEE Trans. on Computers, Vol. 39, No. 8, Aug. 1990.

[18] P. Kornerup, D.W. Matula, *An algorithm for redundant binary bit-pipelined rational arithmetic,* IEEE Trans. on Computers, Vol. 39, No. 8, Aug. 1990.

[19] P.J. Potts with A. Edalat, *Exact real arithmetic using Möbius transformations,* PhD Thesis, Imperial College, London, March 1999.

[20] J.M. Muller, *Elementary Functions, Algorithms and Implementation,* Birkhaeuser, Boston, 1997.

[21] Waterloo Maple Inc., *Maple V,*
http://www.maplesoft.com/.

[22] I. Koren, O. Zinaty, *Evaluating Elementary Functions in a Numerical Coprocessor Based on Rational Approximations,* IEEE Trans. on Computers, Vol. 39, No. 8, Aug. 1990.