Exact Real Computer Arithmetic

Peter John Potts **Abbas Edalat**
Department of Computing
Imperial College
180 Queen's Gate, London SW7 2BZ, United Kingdom
{pjp,ae}@doc.ic.ac.uk

21 March 1997

1 Introduction

Real numbers are usually represented by finite strings of digits belonging to some digit set. The real number representation specifies a function that maps strings to real numbers or real intervals with distinct end-points. For example, IEEE 754 single precision floating point [9] is encoded in 32 binary bits using 1 bit for the sign s, 8 bits for the biased exponent e, and 23 bits for the normalised mantissa m without the leading 1. The basic format represents the real number

$$(-1)^s \, 2^{e-127} \, (1.m).$$

However, finite strings of digits can only represent a limited subset of the real numbers exactly because many real numbers have too many significant digits (such as π or $\sqrt{2}$) or are too large or too small. This means that most real numbers are represented by nearby real numbers or enclosing real intervals with distinct end-points giving rise to the notion of round-off errors. This is generally accepted for a wide range of applications. However, it is well-known that the accumulation of round-off errors due to a large number of calculations can produce grossly inaccurate or even incorrect results. Furthermore, current floating point representations in computers have so many bits that verification of floating point functions cannot be achieved by exhaustively testing every possible input combination.

Interval analysis [17] has been used to partially circumvent this problem by maintaining a pair of bounding numbers that is guaranteed to contain the real number or interval in question. However, this interval can get unjustifiably large and thereby convey very little information.

Alternatively, by allowing infinite strings of digits all the real numbers can be represented exactly. The digits in an infinite string are normally used to construct a sequence of nested real intervals whose lengths converge to zero. The intersection of these intervals is a singleton set whose element is the real number being represented.

Furthermore, basic arithmetic operations are only computable if the representation is *redundant*. In other words, there must be more than one representation for every real number.

In the literature, there are broadly speaking three frameworks for exact real computer arithmetic:

(i) *Infinite sequences of linear maps* proposed by Avizienis [1] and appeared in the work of Watanuki et al [23], Boehm an Cartwright [2], Di Gianantonio [5], Escardo [4], Nielsen et al [18] and Menissier-Morain [16].

(ii) *Continued fraction expansions* proposed by Gosper [7], developed by Peyton Jones [10] and Vuillemin [21] and advanced more recently by Kornerup et al [15, 13, 12, 14].

(iii) *Infinite composition of linear fractional transformations* (also known as homographies or Möbius transformations) generalises the other two frameworks as demonstrated by Vuillemin [21]. Nielsen et al [18] showed that this framework can be used to represent quasi-normalised floating point [23].

We introduce here a new, feasible and incremental representation of the extended real numbers based on the composition of linear fractional transformations with *either all non-negative or all non-positive integer coefficients* [20].

Prototypes have been implemented in C++, Java and Miranda.

2 Linear Fractional Transformations

A natural way to represent a real number, r say, is by a sequence of nested rational intervals $\{[p_n, q_n] : n \in \mathbb{N}\}$ enclosing r such that the sequence of interval lengths converges to zero [8, 17]:

$$[p_0, q_0] \supseteq [p_1, q_1] \supseteq [p_2, q_2] \supseteq [p_3, q_3] \supseteq \cdots$$

Let $\mathbb{R}$ denote the set of real numbers with the Euclidean topology, $\mathbb{R}^\infty$ the one point compactification of $\mathbb{R}$ and $[0, \infty]$ the one point compactification of the non-negative real numbers $[0, \infty)$. The closed intervals $[a, b]$ in $\mathbb{R}^\infty$ are defined as the points from a to b in the numerically increasing direction, possibly including ∞.

Let us make the following convenient definitions:

$$\begin{aligned}
\mathbb{V} &= \left\{ \begin{pmatrix} a \\ b \end{pmatrix} : a \neq 0 \text{ or } b \neq 0 \right\} \\
\mathbb{M} &= \left\{ \begin{pmatrix} a & c \\ b & d \end{pmatrix} : \begin{vmatrix} a & c \\ b & d \end{vmatrix} \neq 0 \right\} \\
\mathbb{T} &= \left\{ \begin{pmatrix} a & c & e & g \\ b & d & f & h \end{pmatrix} : \begin{vmatrix} ax+e & cx+g \\ bx+f & dx+h \end{vmatrix} \text{ and } \begin{vmatrix} ay+c & ey+g \\ by+d & fy+h \end{vmatrix} \text{ are non-trivial} \right\}
\end{aligned}$$

Here $\begin{vmatrix} a & c \\ b & d \end{vmatrix}$ refers to the determinant of the matrix $\begin{pmatrix} a & c \\ b & d \end{pmatrix}$, which is $ad - bc$ of course.

Definition 2.1 *A* 0-dimensional linear fractional transformation (lft) *with real coefficients is a fraction in* $\mathbb{R}^\infty$*, namely a homogeneous coordinate representation of an extended real number,*

$$t_{\begin{pmatrix} a \\ b \end{pmatrix}} = \frac{a}{b} \tag{1}$$

where $\begin{pmatrix} a \\ b \end{pmatrix} \in \mathbb{V}$*. A* 1-dimensional lft *with real coefficients is a function from* $\mathbb{R}^\infty$ *to* $\mathbb{R}^\infty$ *with the general form*

$$t_{\begin{pmatrix} a & c \\ b & d \end{pmatrix}}(x) = \frac{ax + c}{bx + d} \tag{2}$$

where $\begin{pmatrix} a & c \\ b & d \end{pmatrix} \in \mathbb{M}$*. A* 2-dimensional lft *with real coefficients is a function from* $\mathbb{R}^\infty \times \mathbb{R}^\infty$ *to* $\mathbb{R}^\infty$ *with the general form*

$$t_{\begin{pmatrix} a & c & e & g \\ b & d & f & h \end{pmatrix}}(x, y) = \frac{axy + cx + ey + g}{bxy + dx + fy + h} \tag{3}$$

where $\begin{pmatrix} a & c & e & g \\ b & d & f & h \end{pmatrix} \in \mathbb{T}$.

In homogeneous coordinates, a 1-dimensional lft reduces to matrix multiplication

$$\begin{aligned} t_{\begin{pmatrix} a & c \\ b & d \end{pmatrix}} \quad : \quad & \mathbb{R}^\infty \to \mathbb{R}^\infty \\ & \begin{pmatrix} p \\ q \end{pmatrix} \mapsto \begin{pmatrix} ap + cq \\ bp + dq \end{pmatrix}. \end{aligned}$$

Therefore, it is convenient to drop the t in Equations (1), (2) and (3), We will also refer to the coefficients of a 0-dimensional lft as a vector, the coefficients of a 1-dimensional lft as a matrix and the coefficients of a 2-dimensional lft as a tensor.

Definition 2.2 *The* information $\mathsf{Info}(P)$ *contained by an arbitrary lft P is the interval in* $\mathbb{R}^\infty$ *defined by* $\mathsf{Info}(V) = \{V\}$, $\mathsf{Info}(M) = M([0, \infty])$ *and* $\mathsf{Info}(T) = T([0, \infty], [0, \infty])$.

Consider a vector V as a pair of numbers denoted (V_0, V_1),

$$\begin{pmatrix} a \\ b \end{pmatrix} \equiv (a, b)$$

and consider a matrix M as a pair of vectors denoted (M_0, M_1),

$$\begin{aligned}\begin{pmatrix} a & c \\ b & d \end{pmatrix} &\equiv \left(\begin{pmatrix} a \\ b \end{pmatrix}, \begin{pmatrix} c \\ d \end{pmatrix}\right) \\ &\equiv ((a,b),(c,d)),\end{aligned}$$

and consider a tensor T as a pair of matrices denoted (T_0, T_1),

$$\begin{aligned}\begin{pmatrix} a & c & e & g \\ b & d & f & h \end{pmatrix} &\equiv \left(\begin{pmatrix} a & c \\ b & d \end{pmatrix}, \begin{pmatrix} e & g \\ f & h \end{pmatrix}\right) \\ &\equiv \left(\left(\begin{pmatrix} a \\ b \end{pmatrix}, \begin{pmatrix} c \\ d \end{pmatrix}\right), \left(\begin{pmatrix} e \\ f \end{pmatrix}, \begin{pmatrix} g \\ h \end{pmatrix}\right)\right) \\ &\equiv (((a,b),(c,d)),((e,f),(g,h))).\end{aligned}$$

The first important observation is that $\mathsf{Info}(P)$ is computationally easy to evaluate. The endpoints of the interval $\mathsf{Info}(P)$ (provided it has any) correspond to a particular pair of the vectors that make up the coefficients. For a matrix $M = \begin{pmatrix} a & c \\ b & d \end{pmatrix}$, we have

$$\mathsf{Info}(M) = \begin{cases} \left[\frac{a}{b}, \frac{c}{d}\right] & \text{if } \mathsf{det}(M) < 0 \\ \left[\frac{c}{d}, \frac{a}{b}\right] & \text{if } \mathsf{det}(M) > 0 \end{cases}$$

and for a tensor $T = \begin{pmatrix} a & c & e & g \\ b & d & f & h \end{pmatrix}$, we have

$$\mathsf{Info}(T) = \mathsf{Info}\left(\begin{pmatrix} a & c \\ b & d \end{pmatrix}\right) \bigcup \mathsf{Info}\left(\begin{pmatrix} e & g \\ f & h \end{pmatrix}\right) \bigcup \mathsf{Info}\left(\begin{pmatrix} a & e \\ b & f \end{pmatrix}\right) \bigcup \mathsf{Info}\left(\begin{pmatrix} c & g \\ d & h \end{pmatrix}\right).$$

Definition 2.3 *An arbitrary lft P satisfies the* refinement property*, denoted $\mathcal{R}(P)$, if the coefficients of P are all non-negative or all non-positive.*

Let us define $\mathbb{V}^+ = \{V \in \mathbb{V} : \mathcal{R}(V)\}$, $\mathbb{M}^+ = \{M \in \mathbb{M} : \mathcal{R}(M)\}$ and $\mathbb{T}^+ = \{T \in \mathbb{T} : \mathcal{R}(T)\}$.

The second important observation is that $\mathsf{Info}(P) \subseteq [0, \infty]$ if and only if P satisfies the refinement property.

Therefore, the composition $Q \circ P$ of arbitrary lft's P and Q where P satisfies the refinement property corresponds to interval refinement. This is because, in this case, $\mathsf{Info}(Q \circ P) \subseteq \mathsf{Info}(Q)$. In other words, P refines the information given by Q.

3 Normal Products

Proposition 3.1 *Given two rational intervals $I = \mathsf{Info}(M)$ and $J = \mathsf{Info}(N)$ represented by the two matrices M and N, then $I \subseteq J$ if and only if there exists a matrix K with integer coefficients satisfying the refinement property such that $M = NK$.*

Therefore any real number can be represented as the intersection

$$\bigcap_{n \geq 0} M_0 M_1 M_2 \ldots M_n([0, \infty])$$

for a sequence of matrices M_n where M_n satisfies the refinement property for $n \geq 1$. We can denote this real number by an infinite product of matrices

$$\begin{pmatrix} a_0 & c_0 \\ b_0 & d_0 \end{pmatrix} : \begin{pmatrix} a_1 & c_1 \\ b_1 & d_1 \end{pmatrix} : \begin{pmatrix} a_2 & c_2 \\ b_2 & d_2 \end{pmatrix} : \cdots \tag{4}$$

where $M_n = \begin{pmatrix} a_n & c_n \\ b_n & d_n \end{pmatrix}$ and $M_0 \in \mathbb{M}$ and $M_n \in \mathbb{M}^+$ with $n \geq 1$. We will call this an *infinite normal product.* Notice however that an infinite normal product does not in general represent a point, although it always represents an interval. However, we are only interested in infinite normal products that converge to a point. We will call the first matrix M_0 the *sign matrix* and the others matrices M_n with $n \geq 1$ we will call the *digit matrices.* A singular matrix is in fact a constant that can be replaced by a vector, thus terminating the product; this we will call a *finite normal product.* Thus, a finite normal product represents a rational number, whereas an infinite normal product may represent any number.

The characteristic of a normal product is that the sign matrix identifies the interval in $\mathbb{R}^\infty$ of interest, while the digit matrices refine this interval to a point. Therefore, we can consider two types called *signed normal products (snp)* and *unsigned normal products (unp).* So, we can think of a signed normal product as a sign matrix followed by an unsigned normal product. Additionally, an unsigned normal product consists of a digit matrix followed by an unsigned normal products recursively. So, signed normal products represent real numbers in $\mathbb{R}^\infty$, while unsigned normal products represent real numbers in $[0, \infty]$. In BNF, we have

$$\begin{array}{rcll} \mathrm{snp}_0 & ::= & \alpha \mid \beta : \mathrm{unp}_{\phi(0,\beta)} & \text{where } \alpha \in \Phi_v(0) \text{ and } \beta \in \Phi_m(0) \\ \mathrm{unp}_\tau & ::= & \gamma \mid \delta : \mathrm{unp}_{\phi(\tau,\delta)} & \text{where } \gamma \in \Phi_v(\tau) \text{ and } \delta \in \Phi_m(\tau) \text{ and } \tau \geq 1. \end{array}$$

We call $\Phi_v(\tau)$ and $\Phi_m(\tau)$ the *state sets.* They contain the next allowable vectors and matrices in the normal product given that we are in state τ. The initial state is 0. Therefore $\Phi_v(0)$ and $\Phi_m(0)$ contain the allowable sign vectors and matrices. $\phi(\tau, M)$ is called the *state transition function* and specifies the next state given that the previous state is τ and the previous matrix in the normal product is M. Taken together, the state sets and the state transition function restrict the allowable sequences of vectors and matrices in a normal product.

For the *most general normal product* we define the *state sets* to be

$$\begin{aligned} \Phi_v(0) &= \mathbb{V} \\ \Phi_v(1) &= \mathbb{V}^+ \\ \Phi_m(0) &= \mathbb{M} \\ \Phi_m(1) &= \mathbb{M}^+ \end{aligned}$$

and the *state transition function* $\phi(\tau, M)$ by

$\phi(\tau, M)$	$\tau = 0$	$\tau = 1$
$M \in \Phi_m(0)$	1	
$M \in \Phi_m(1)$		1

This notation may seem excessive at this point, but it will prove useful when we define various notions of *exact floating point* later.

4 Expression Trees

Let us generalise normal products even further to *expression trees* by including tensors. We can do this by constructing *signed expression trees (sexp)* and *unsigned expression trees (uexp)* according to the BNF prescription

$$\begin{aligned} \text{sexp} &::= \mathbb{V} \mid \mathbb{M}(\text{uexp}) \mid \mathbb{T}(\text{uexp}, \text{uexp}) \\ \text{uexp} &::= \mathbb{V}^+ \mid \mathbb{M}^+(\text{uexp}) \mid \mathbb{T}^+(\text{uexp}, \text{uexp}). \end{aligned}$$

If an expression E is a vector then it simply represents a rational number.

If an expression E is a matrix M applied to another expression E' then it represents an interval which is a subset of $\mathsf{Info}(M)$. The information in the matrix M can be refined by incorporating information from the expression E' according to equations (5) and (6). We call this process *matrix absorption.*

If an expression E is a tensor T applied to two other expressions E' and E'' then it represents an interval which is a subset of $\mathsf{Info}(T)$. Again, the information in the tensor T can be refined by incorporating information from the expressions E' and E'' according to the equations (7) to (10). We call this process *tensor absorption.*

What happens if we meet a tensor in a subexpression? Absorption equations could be devised to cater for this scenario. However, this would lead to higher dimensional lft's and increased complexity. A solution is to force subexpression tensors to output information in the form of matrices that reflect the information contained in them according to equation (11). We call this process *tensor emission*

5 Absorption and Emission Equations

In order to simplify composition of lft's of various dimensions, we define the *dot product*, the *left product* and the *right product*, denoted respectively by $\cdot$, Ⓛ and Ⓡ as follows:

$$\begin{aligned} (M \cdot V)_i &= \sum_{j=0,1} M_{ij} V_j \\ (M \cdot N) &= (M \cdot N_0, M \cdot N_1) \\ (M \cdot T) &= (M \cdot T_0, M \cdot T_1) \end{aligned}$$

$$\begin{aligned} T \circledR V &= (T_0 \cdot V, T_1 \cdot V) \\ T \circledR M &= (T_0 \cdot M, T_1 \cdot M) \\ T \circledL V &= T^{\mathsf{T}} \circledR V \\ T \circledL M &= (T^{\mathsf{T}} \circledR M)^{\mathsf{T}} \end{aligned}$$

where T^{T} indicates the transpose of T defined by swapping its middle two columns. Note that dot product is just conventional matrix multiplication. Let us also define the *mediant* of a matrix by

$$\overline{\begin{pmatrix} a & c \\ b & d \end{pmatrix}} = \begin{pmatrix} a+c \\ b+d \end{pmatrix}.$$

Proposition 5.1 *The following* matrix absorption *equations hold:*

$$M(V) = M \cdot V \tag{5}$$

$$M(N(x)) = (M \cdot N)(x) \tag{6}$$

The following tensor absorption *equations hold:*

$$T(V, y) = \begin{cases} \overline{T \circledL V} & \textit{if } |T \circledL V| = 0 \\ (T \circledL V)(y) & \textit{if } |T \circledL V| \neq 0 \end{cases} \tag{7}$$

$$T(M(x), y) = (T \circledL M)(x, y) \tag{8}$$

$$T(x, V) = \begin{cases} \overline{T \circledR V} & \textit{if } |T \circledR V| = 0 \\ (T \circledR V)(x) & \textit{if } |T \circledR V| \neq 0 \end{cases} \tag{9}$$

$$T(x, M(y)) = (T \circledR M)(x, y). \tag{10}$$

Proof Here is the proof for one of the absorption equations:

$$\begin{aligned} T(x, V) &= \begin{pmatrix} a & c & e & g \\ b & d & f & h \end{pmatrix} \left(x, \begin{pmatrix} i \\ j \end{pmatrix} \right) \\ &= \frac{\frac{aix}{j} + cx + \frac{ei}{j} + g}{\frac{bix}{j} + dx + \frac{fi}{j} + h} \\ &= \frac{aix + cjx + ei + gj}{bix + djx + fi + hj} \\ &= \begin{pmatrix} ai + cj & ei + gj \\ bi + dj & fi + hj \end{pmatrix} (x) \\ &= \left(\begin{pmatrix} a & c \\ b & d \end{pmatrix} \cdot \begin{pmatrix} i \\ j \end{pmatrix}, \begin{pmatrix} e & g \\ f & h \end{pmatrix} \cdot \begin{pmatrix} i \\ j \end{pmatrix} \right) (x) \\ &= (T_0 \cdot V, T_1 \cdot V)(x) \\ &= (T \circledR V)(x) \\ &= \begin{cases} \overline{T \circledR V} & \text{if } |T \circledR V| = 0 \\ (T \circledR V)(x) & \text{if } |T \circledR V| \neq 0 \end{cases} \end{aligned}$$

Note that the left and right products of a tensor with a vector may give a singular matrix, which is essentially a vector given by it's mediant. $\square$

The third important observation is that given a matrix M and an arbitrary lft P, $\mathsf{Info}(M) \supseteq \mathsf{Info}(P)$ if and only if $\mathcal{R}(M^{-1} \cdot P)$ where M^{-1} is the matrix inverse of M. The scaling invariance of lft's means that we can simplify the definition of the matrix inverse to

$$\begin{pmatrix} a & c \\ b & d \end{pmatrix}^{-1} = \begin{pmatrix} d & -c \\ -b & a \end{pmatrix}.$$

Proposition 5.2 *The following* emission *equations hold:*

$$\begin{array}{rcll} V & = & E(E^{-1} \cdot V) & \textit{if } \mathsf{Info}(E) \supseteq \mathsf{Info}(V) \\ M(x) & = & E((E^{-1} \cdot M)(x)) & \textit{if } \mathsf{Info}(E) \supseteq \mathsf{Info}(M) \\ T(x,y) & = & E((E^{-1} \cdot T)(x,y)) & \textit{if } \mathsf{Info}(E) \supseteq \mathsf{Info}(T) \end{array} \tag{11}$$

The conditions are necessary in order to ensure that $(E^{-1} \cdot V)$, $(E^{-1} \cdot M)$ and $(E^{-1} \cdot T)$ satisfy the refinement property.

Note that the fundamental aspects of these absorption and emission equations are essentially well known [18]. The novel aspects include the introduction of vectors and, very importantly, the repeated insistence on lft's satisfying the refinement property leading to a computationally efficient test for interval inclusion.

6 Tensor Absorption Strategy

For computing the value of $T(x,y)$, we need a **strategy** for deciding whether to absorb from x (*left absorption*) or from y (*right absorption*). All we know about x and y is that they are non-negative real numbers. So, what we need is a function $\mathsf{strategy}(T)$ which evaluates to **left**, **right** or **either**. By convention, we choose left absorption when we have a free choice. This enables algorithms to be made in the knowledge that there is a preferred absorption direction. See the $\mathsf{tan}(x)$ example in section 12.

6.1 The Outcome Minimisation Strategy

One approach is to minimise the possible range of outcomes. The range of outcomes for a left absorption is given by

$$L(T) = \sup \left\{ d\left(\frac{ax+e}{bx+f}, \frac{cx+g}{dx+h}\right) : x \in [0,\infty] \right\}$$

and the range of outcomes for a right absorption is given by

$$R(T) = \sup \left\{ d\left(\frac{ay+c}{by+d}, \frac{ey+g}{fy+h}\right) : y \in [0,\infty] \right\}$$

where $d(x,y)$ is a suitable metric such as

$$d(x,y) = \left| \frac{x}{x+1} - \frac{y}{y+1} \right|.$$

The exact formula involves a square root. However, a good approximation in practice for $L(T)$ and $R(T)$, which is exact in many cases, is given by

$$\begin{aligned} L(T) &\approx \mathsf{width}(J \cdot T^{\mathsf{T}}) \\ R(T) &\approx \mathsf{width}(J \cdot T) \end{aligned}$$

where

$$\begin{aligned} J &= \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \\ \mathsf{width}(T) &= \max\{|T_0(0) - T_1(0)|, |T_0(\infty) - T_1(\infty)|\}. \end{aligned}$$

So, the strategy is:

$$\mathsf{strategy}(T) = \begin{cases} \textbf{left} & \text{if } L(T) < R(T) \\ \textbf{either} & \text{if } L(T) = R(T) \\ \textbf{right} & \text{if } L(T) > R(T) \end{cases}$$

6.2 The Information Overlap Strategy

An alternative approach is to consider the pairs of matrices in tensor T and the pairs of matrices in tensor T^{T}.

During left absorption the information in each of the matrices in tensor T^{T} is refined, while during right absorption the information in each of the matrices in tensor T is refined.

Absorption is more effective if the pair of matrices under consideration have overlapping information because then the overall information refinement for the tensor is likely to be more dramatic on average. A more important observation is that absorption eventually leads to an empty intersection of information thus ensuring that the strategy is fair.

Let us define the function $\mathsf{overlap}(T)$ by

$$\begin{aligned} \mathsf{overlap} : \mathbb{T} &\to \text{boolean} \\ T &\mapsto \mathsf{Info}(T_0) \cap \mathsf{Info}(T_1) \neq \emptyset \end{aligned}$$

Given $T = \begin{pmatrix} a & c & e & g \\ b & d & f & h \end{pmatrix}$, it can be shown that the boolean expression

$$\mathsf{Info}(T_0) \cap \mathsf{Info}(T_1) = \emptyset$$

is equivalent to

$$\mathsf{Info}\left(\begin{pmatrix} d & -c \\ b & -a \end{pmatrix} \cdot \begin{pmatrix} e & g \\ f & h \end{pmatrix}\right) \subseteq (0, \infty).$$

It is always the case that at least one of $\mathsf{overlap}(T)$ and $\mathsf{overlap}(T^{\mathsf{T}})$ is true.

Therefore, a straightforward strategy is:

$$\mathsf{strategy}(T) = \begin{cases} \textbf{left} & \text{if not } \mathsf{overlap}(T) \\ \textbf{either} & \text{if } \mathsf{overlap}(T) \text{ and } \mathsf{overlap}(T^{\mathsf{T}}) \\ \textbf{right} & \text{if not } \mathsf{overlap}(T^{\mathsf{T}}) \end{cases}$$

7 Unbiased Exact Floating Point

Storing a real number as a general normal product may be efficient in certain cases, but in general we need to ensure that the digit matrices in an infinite normal product refine the sign matrix at a steady rate. Also, we need a way to control the size of the integer coefficients in tensors. Both these objectives can be achieved by restricting the state sets.

In particular, we put forward the following important representation that we shall call *base B unbiased exact floating point*

$$S_\sigma : D^B_{d_1} : D^B_{d_2} : D^B_{d_3} : \cdots$$

where

$$\begin{aligned}
\sigma &\in \mathbb{S} = \{+, \infty, -, 0\} \\
d_i &\in \mathbb{D} = \{n - B, B - n : n \in \mathbb{N} \cap [1, B]\} \\
S_+ &= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \\
S_\infty &= \begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix} \\
S_- &= \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \\
S_0 &= \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix} \\
D^B_d &= \begin{pmatrix} B + d + 1 & B + d - 1 \\ B - d - 1 & B - d + 1 \end{pmatrix}.
\end{aligned}$$

Therefore S_σ is the *sign matrix* while $D^B_{d_i}$ for $i \geq 1$ make up the *digit matrices* in *base B*. If the digit matrices begin with a block of n D^B_{1-B}'s or D^B_{B-1}'s, we call this the *exponent* because it indicates the order of magnitude of the real number in question. The rest of the digit matrices is called the *mantissa.*

The sign matrices have been chosen such that they divide $\mathbb{R}^\infty$ in a natural, unbiased and redundant manner; $\mathsf{Info}(S_+) = [0, \infty]$, $\mathsf{Info}(S_\infty) = [1, -1]$, $\mathsf{Info}(S_-) = [\infty, 0]$ and $\mathsf{Info}(S_0) = [-1, 1]$. Recall that redundancy is important for computability. Also, they have positive determinants, which is convenient for implementation, and form a cyclic group (up to scaling invariance)

$$\begin{aligned}
S^0_\infty &= S_+ \\
S^1_\infty &= S_\infty \\
S^2_\infty &= S_- \\
S^3_\infty &= S_0 \\
S^4_\infty &= S^0_\infty,
\end{aligned}$$

which is mathematically useful for deriving algorithms.

The digit matrices satisfy the following property, which is reminiscent of redundant sequences of digits proposed by Avizienis [1].

$$D^B_{d_1} \cdot D^B_{d_2} \cdot \ldots \cdot D^B_{d_n} = \begin{pmatrix} B^n + m + 1 & B^n + m - 1 \\ B^n - m - 1 & B^n - m + 1 \end{pmatrix}$$

where

$$m = \sum_{i=1}^{n} d_i B^{n-i}.$$

In fact, the sign matrix S_0 followed by $D^2_{d_1} : D^2_{d_2} : D^2_{d_3} : \cdots$ corresponds exactly to redundant sequences of binary digits as pointed out by Nielsen et al [18]. The novelty here is that we extend the representation to any base B and to the other three signs; $+$, ∞ and $-$. For S_0, we have

$$\begin{aligned} S_0 D^B_{d_1} \cdot D^B_{d_2} \cdot \ldots \cdot D^B_{d_n} &= \begin{pmatrix} m+1 & m-1 \\ B^n & B^n \end{pmatrix} \\ \mathsf{Info}(S_0 D^B_{d_1} \cdot D^B_{d_2} \cdot \ldots \cdot D^B_{d_n}) &= \left[\frac{m-1}{B^n}, \frac{m+1}{B^n}\right]. \end{aligned} \tag{12}$$

The origin of D^B_d becomes clearer when we consider the following commutative diagram:

$$\begin{tikzcd} {[0,\infty]} \arrow[r, "D^B_d"] \arrow[d, "S_0"'] & {[0,\infty]} \arrow[d, "S_0"] \\ {[-1,1]} \arrow[r, "R^B_d"] & {[-1,1]} \end{tikzcd}$$

where

$$\begin{aligned} R^B_d &= S_0 \cdot D^B_d \cdot S_0^{-1} \\ &= \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} B+d+1 & B+d-1 \\ B-d-1 & B-d+1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix} \\ &= \begin{pmatrix} 1 & d \\ 0 & B \end{pmatrix} \\ R^B_d(x) &= \frac{x+d}{B} \end{aligned}$$

So, R^B_d are the standard affine maps corresponding to the digits in the the representation of the real numbers over the interval $[-1,1]$ by redundant sequences of digits. In particular, R^2_{-1} maps $[-1,1]$ to the left half $[-1,0]$, R^2_0 maps $[-1,1]$ to the middle half $[-\frac{1}{2}, \frac{1}{2}]$ and R^2_1 maps $[-1,1]$ to the right half $[0,1]$.

So, the state sets for unbiased exact floating point are

$$\begin{aligned}
\Phi_v(0) &= \emptyset \\
\Phi_v(1) &= \emptyset \\
\Phi_m(0) &= \{S_\sigma : \sigma \in \mathbb{S}\} \\
\Phi_m(1) &= \{D_n^B : n \in \mathbb{D}\}
\end{aligned}$$

and the state transition function $\phi(\tau, M)$ is given by

$\phi(\tau, M)$	$\tau = 0$	$\tau = 1$
$M \in \Phi_m(0)$	1	
$M \in \Phi_m(1)$		1

A useful hybrid between general normal products and unbiased exact floating point involves setting $\Phi_v(0)$ and $\Phi_v(1)$ to

$$\begin{aligned}
\Phi_v(0) &= \left\{ \begin{pmatrix} a \\ b \end{pmatrix} : \begin{pmatrix} a \\ b \end{pmatrix} \in \mathbb{V} \text{ and } |a|, |b| < \Omega \right\} \\
\Phi_v(1) &= \left\{ \begin{pmatrix} a \\ b \end{pmatrix} : \begin{pmatrix} a \\ b \end{pmatrix} \in \mathbb{V}^+ \text{ and } |a|, |b| < \Omega \right\}
\end{aligned}$$

where Ω may correspond to the maximum integer that fits into a computer word.

Interesting bases include 2 with digits $\{-1, 0, 1\}$ and the golden ratio $\phi = \frac{1+\sqrt{5}}{2}$ with digits $\{1 - \phi, \phi - 1\}$ [19, 6].

This representation of the real numbers leads to efficient algorithms for a wide range of operations including the transcendental functions. The only exception is for addition and subtraction. In which case, we need biased exact floating point.

8 Biased Exact Floating Point

Biased exact floating point leads to efficient algorithms for addition and subtraction that are essentially the same as those for redundant sequences of digits with the added benefit of incrementality even at the exponent level. This means that infinity can be handled as if it were any other real number.

We restrict the allowable sequences of matrices as follows

$$\left\{ \begin{array}{lcl}
S_+ & : & \left\{ \begin{array}{lcl} L & & \\ H & : & X^* : F_n \end{array} \right\} \\
S_\infty & : & Z^* : \left\{ \begin{array}{l} I : X^* : F_n \\ \hat{I} : \hat{X}^* : \hat{F}_n \end{array} \right\} \\
S_- & : & \left\{ \begin{array}{lcl} \hat{L} & & \\ \hat{H} & : & \hat{X}^* : \hat{F}_n \end{array} \right\} \\
S_0 & &
\end{array} \right\} : \left\{ \begin{array}{c} D_{1-B}^B \\ \vdots \\ D_{B-1}^B \end{array} \right\}^*$$

where M^* indicates zero or more occurrences of M and $\left\{ \begin{array}{c} P \\ Q \end{array} \right\}$ indicates a choice between the sequences P and Q and

$$\begin{aligned}
n &\in \mathbb{F} = \mathbb{N} \cap [1, B) \\
L &= \begin{pmatrix} 2 & 0 \\ 1 & 1 \end{pmatrix} \\
H &= \begin{pmatrix} B & 1 \\ 0 & 1 \end{pmatrix} \\
X &= \begin{pmatrix} B^2 & B-1 \\ 0 & B \end{pmatrix} \\
F_n &= \begin{pmatrix} n+1 & n-1 \\ B & B \end{pmatrix} \\
I &= \begin{pmatrix} B & 0 \\ B & 2 \end{pmatrix} \\
Z &= \begin{pmatrix} B+1 & B-1 \\ B-1 & B+1 \end{pmatrix} \\
\widehat{\begin{pmatrix} a & c \\ b & d \end{pmatrix}} &= \begin{pmatrix} d & b \\ c & a \end{pmatrix}.
\end{aligned}$$

These matrices are such that once we reach the matrices D^B_{1-B} to D^B_{B-1} the nested intervals correspond to conventional B-adic intervals (as in equation (12)). The following properties can be easily derived:

$$\begin{aligned}
\mathsf{Info}(S_+ L) &= [0, 2] \\
\mathsf{Info}(S_+ H X^e) &= [B^e, \infty] \\
\mathsf{Info}(S_+ H X^e F_n) &= [nB^e, (n+2)B^e] \\
\mathsf{Info}(S_\infty Z^e) &= [B^e, -B^e] \\
\mathsf{Info}(S_\infty Z^e I) &= [B^e, \infty] \\
\mathsf{Info}(S_\infty Z^e \hat{I}) &= [\infty, -B^e] \\
\mathsf{Info}(S_- \hat{L}) &= [-2, 0] \\
\mathsf{Info}(S_- \hat{H} \hat{X}^e) &= [\infty, -B^e] \\
\mathsf{Info}(S_- \hat{H} \hat{X}^e \hat{F}_n) &= [-(n+2)B^e, -nB^e].
\end{aligned}$$

It should be pointed out that Nielsen et al [18] considered a similar idea for quasi-normalised floating point. However, we restrict the digit matrices to ones satisfying the refinement property. This means that our algorithms for interval inclusion are much more efficient. Also, our representation includes three types of infinity; namely positive infinity, negative infinity and an infinity with an unknown sign. This means that an algorithm for addition can emit matrices without necessarily first absorbing the entire exponent part of the two arguments.

To summerise, the state sets are

$$\begin{aligned}
\Phi_v(n) &= \emptyset \\
\Phi_m(0) &= \{S_\sigma : \sigma \in \mathbb{S}\} \\
\Phi_m(1) &= \{D^B_n : n \in \mathbb{D}\} \\
\Phi_m(2) &= \{L, H\} \\
\Phi_m(3) &= \{I, Z, \hat{I}\} \\
\Phi_m(4) &= \{\hat{L}, \hat{H}\} \\
\Phi_m(5) &= \{X\} \cup \{F_n : n \in \mathbb{F}\} \\
\Phi_m(6) &= \{\hat{X}\} \cup \{\hat{F}_n : n \in \mathbb{F}\}
\end{aligned}$$

and the state transition function $\phi(\tau, M)$ is given by

$\phi(\tau, M)$	$\tau = 0$	$\tau = 1$	$\tau = 2$	$\tau = 3$	$\tau = 4$	$\tau = 5$	$\tau = 6$
$M = S_+$	2						
$M = S_\infty$	3						
$M = S_-$	4						
$M = S_0$	1						
$M = D^B_n$		1					
$M = L$			1				
$M = H$			5				
$M = I$				5			
$M = Z$				3			
$M = \hat{I}$				6			
$M = \hat{L}$					1		
$M = \hat{H}$					6		
$M = X$						5	
$M = F_n$						1	
$M = \hat{X}$							6
$M = \hat{F}_n$							1

9 Normalisation Algorithms

In this section, we describe a basic set of lazy functions that allow signed normal products to be generated from

(0-dimensional lft) rational numbers,

(1-dimensional lft) matrices applied to signed general normal products and

(2-dimensional lft) tensors applied to signed general normal products.

The signed normal products generated follow the restrictions specified by the state sets $\Phi_v(\tau)$ and $\Phi_m(\tau)$ for $\tau \in \mathbb{N}$ and the state transition function $\phi(\tau, M)$ for $\tau \in \mathbb{N}$ and $M \in \mathbb{M}$.

9.1 Vector Algorithm

The expression $\mathsf{svector}_0(V)$ evaluates the vector $V \in \mathbb{V}$ to a signed normal product.

$$\begin{aligned} \mathsf{svector}_0 : \mathbb{V} &\rightarrow \mathrm{snp}_0 \\ V &\mapsto V \text{ if } V \in \Phi_v(0) \\ V &\mapsto S : \mathsf{uvector}_{\phi(0,S)}(S^{-1} \cdot V) \\ &\qquad \text{if } \exists S \in \Phi_m(0) \text{ such that } \mathsf{Info}(S) \supseteq \mathsf{Info}(V) \end{aligned}$$

In words, if V is an allowable sign vector then it is left unchanged otherwise find a sign matrix S that can be emitted and continue with the remainder.

The expression $\mathsf{uvector}_\tau(V)$ evaluates the vector $V \in \mathbb{V}^+$ to a state τ unsigned normal product.

$$\begin{aligned} \mathsf{uvector}_\tau : \mathbb{V}^+ &\rightarrow \mathrm{unp}_\tau \\ V &\mapsto V \text{ if } V \in \Phi_v(\tau) \\ V &\mapsto D : \mathsf{uvector}_{\phi(\tau,D)}(D^{-1} \cdot V) \\ &\qquad \text{if } \exists D \in \Phi_m(\tau) \text{ such that } \mathsf{Info}(D) \supseteq \mathsf{Info}(V) \end{aligned}$$

In words, if V is an allowable digit vector in state τ then leave it unchanged otherwise find a digit matrix D that can be emitted in this state and continue with the remainder.

9.2 Matrix Algorithm

The expression $\mathsf{smatrixs}_0(M, E_x)$ evaluates the matrix $M \in \mathbb{M}$ and the signed general normal product E_x to a signed normal product.

$$\begin{aligned} \mathsf{smatrixs}_0 : \mathbb{M} \times \mathrm{sexp} &\rightarrow \mathrm{snp}_0 \\ (M, V_x) &\mapsto \mathsf{svector}_0(M \cdot V_x) \\ (M, S_x : U_x) &\mapsto \mathsf{smatrixu}_0(M \cdot S_x, U_x) \end{aligned}$$

In words, the signed part of the signed general normal product E_x must be absorbed into M before any further steps can be taken.

The expression $\mathsf{smatrixu}_0(M, E_x)$ evaluates the matrix $M \in \mathbb{M}$ and the unsigned general normal product E_x to a signed normal product.

$$\mathsf{smatrixu}_0 : \mathbb{M} \times \mathrm{uexp} \rightarrow \mathrm{snp}_0$$

$$
\begin{array}{rcl}
(M, V_x) & \mapsto & M : \mathsf{uvector}_{\phi(0,M)}(V_x) \text{ if } M \in \Phi_m(0) \\
(M, D_x : U_x) & \mapsto & M : \mathsf{umatrixu}_{\phi(0,M)}(D_x, U_x) \text{ if } M \in \Phi_m(0) \\
(M, E_x) & \mapsto & S : \mathsf{umatrixu}_{\phi(0,S)}(S^{-1} \cdot M, E_x) \\
& & \text{if } \exists S \in \Phi_m(0) \text{ such that } \mathsf{Info}(S) \supseteq \mathsf{Info}(M) \\
(M, V_x) & \mapsto & \mathsf{svector}_0(M \cdot V_x) \\
(M, D_x : U_x) & \mapsto & \mathsf{smatrixu}_0(M \cdot D_x, U_x)
\end{array}
$$

In words, if M is an allowable sign matrix then it is left unchanged otherwise try to find a sign matrix S that can be emitted and continue with the remainder. Otherwise, more information must be absorbed from E_x.

The expression $\mathsf{umatrixu}_\tau(M, E_x)$ evaluates the matrix $M \in \mathbb{M}^+$ and the unsigned general normal product E_x to a state τ unsigned normal product.

$$
\begin{array}{rcl}
\mathsf{umatrixu}_\tau : \mathbb{M}^+ \times \text{uexp} & \rightarrow & \text{unp}_\tau \\
(M, V_x) & \mapsto & M : \mathsf{uvector}_{\phi(\tau,M)}(V_x) \text{ if } M \in \Phi_m(\tau) \\
(M, D_x : U_x) & \mapsto & M : \mathsf{umatrixu}_{\phi(\tau,M)}(D_x, U_x) \text{ if } M \in \Phi_m(\tau) \\
(M, E_x) & \mapsto & D : \mathsf{umatrixu}_{\phi(\tau,D)}(D^{-1} \cdot M, E_x) \\
& & \text{if } \exists D \in \Phi_m(\tau) \text{ such that } \mathsf{Info}(D) \supseteq \mathsf{Info}(M) \\
(M, V_x) & \mapsto & \mathsf{uvector}_\tau(M \cdot V_x) \\
(M, D_x : U_x) & \mapsto & \mathsf{umatrixu}_\tau(M \cdot D_x, U_x)
\end{array}
$$

In words, if M is an allowable digit matrix in state τ then it is left unchanged otherwise try to find a digit matrix D that can be emitted in this state and continue with the remainder. Otherwise, more information must be absorbed from E_x.

9.3 Tensor Algorithm

The expression $\mathsf{stensors}_0(T, E_x, E_y)$ evaluates the tensor $T \in \mathbb{T}$ and the signed general normal products E_x and E_y to a signed normal product.

$$
\begin{array}{rcl}
\mathsf{stensors}_0 : \mathbb{T} \times \text{sexp} \times \text{sexp} & \rightarrow & \text{snp}_0 \\
(T, V_x, V_y) & \mapsto & \begin{cases} \mathsf{svector}_0(\overline{T Ⓛ V_x}) & \text{if } |T Ⓛ V_x| = 0 \\ \mathsf{svector}_0(T Ⓛ V_x \cdot V_y) & \text{if } |T Ⓛ V_x| \neq 0 \end{cases} \\
(T, V_x, S_y : U_y) & \mapsto & \begin{cases} \mathsf{svector}_0(\overline{T Ⓛ V_x}) & \text{if } |T Ⓛ V_x| = 0 \\ \mathsf{smatrixu}_0(T Ⓛ V_x \cdot S_y, U_y) & \text{if } |T Ⓛ V_x| \neq 0 \end{cases} \\
(T, S_x : U_x, V_y) & \mapsto & \begin{cases} \mathsf{svector}_0(\overline{T Ⓡ V_y}) & \text{if } |T Ⓡ V_y| = 0 \\ \mathsf{smatrixu}_0(T Ⓡ V_y \cdot S_x, U_x) & \text{if } |T Ⓡ V_y| \neq 0 \end{cases} \\
(T, S_x : U_x, S_y : U_y) & \mapsto & \mathsf{stensoru}_0(T Ⓛ S_x Ⓡ S_y, U_x, U_y)
\end{array}
$$

In words, the signed part of the signed general normal products E_x and E_y must be absorbed into T before any further steps can be taken.

The expression $\mathsf{stensoru}_0(T, E_x, E_y)$ evaluates the tensor $T \in \mathbb{T}$ and the unsigned general normal products E_x and E_y to a signed normal product.

$$
\begin{array}{rcl}
\mathsf{stensoru}_0 : \mathbb{T} \times \text{uexp} \times \text{uexp} & \rightarrow & \text{snp}_0 \\
(T, E_x, E_y) & \mapsto & S : \mathsf{utensoru}_{\phi(0,S)}(S^{-1} \cdot T, E_x, E_y) \\
& & \text{if } \exists S \in \Phi_m(0) \text{ such that } \mathsf{Info}(S) \supseteq \mathsf{Info}(T) \\
\text{if } \mathsf{strategy}(T) \neq \textbf{right} \text{ then} & & \\
(T, V_x, U_y) & \mapsto & \begin{cases} \mathsf{svector}_0(\overline{T \textcircled{\tiny L}\, V_x}) & \text{if } |T \textcircled{\tiny L}\, V_x| = 0 \\ \mathsf{smatrixu}_0(T \textcircled{\tiny L}\, V_x, U_y) & \text{if } |T \textcircled{\tiny L}\, V_x| \neq 0 \end{cases} \\
(T, D_x : U_x, U_y) & \mapsto & \mathsf{stensoru}_0(T \textcircled{\tiny L}\, D_x, U_x, U_y) \\
\text{if } \mathsf{strategy}(T) = \textbf{right} \text{ then} & & \\
(T, U_x, V_y) & \mapsto & \begin{cases} \mathsf{svector}_0(\overline{T \textcircled{\tiny R}\, V_y}) & \text{if } |T \textcircled{\tiny R}\, V_y| = 0 \\ \mathsf{smatrixu}_0(T \textcircled{\tiny R}\, V_y, U_x) & \text{if } |T \textcircled{\tiny R}\, V_y| \neq 0 \end{cases} \\
(T, U_x, D_y : U_y) & \mapsto & \mathsf{stensoru}_0(T \textcircled{\tiny R}\, D_y, U_x, U_y)
\end{array}
$$

In words, try to find a sign matrix S that can be emitted and continue with the remainder. Otherwise, more information must be absorbed from E_x or E_y according to the value of $\mathsf{strategy}(T)$. If $\mathsf{strategy}(T) = \textbf{right}$ then absorb from E_y otherwise absorb from E_x.

The expression $\mathsf{utensoru}_\tau(T, E_x, E_y)$ evaluates the tensor $T \in \mathbb{T}^+$ and the unsigned general normal products E_x and E_y to a state τ unsigned normal product.

$$
\begin{array}{rcl}
\mathsf{utensoru}_\tau : \mathbb{T}^+ \times \text{uexp} \times \text{uexp} & \rightarrow & \text{unp}_\tau \\
(T, E_x, E_y) & \mapsto & D : \mathsf{utensoru}_{\phi(\tau,D)}(D^{-1} \cdot T, E_x, E_y) \\
& & \text{if } \exists D \in \Phi_m(x) \text{ such that } \mathsf{Info}(D) \supseteq \mathsf{Info}(T) \\
\text{if } \mathsf{strategy}(T) \neq \textbf{right} \text{ then} & & \\
(T, V_x, U_y) & \mapsto & \begin{cases} \mathsf{uvector}_\tau(\overline{T \textcircled{\tiny L}\, V_x}) & \text{if } |T \textcircled{\tiny R}\, V_x| = 0 \\ \mathsf{umatrixu}_\tau(T \textcircled{\tiny L}\, V_x, U_y) & \text{if } |T \textcircled{\tiny R}\, V_x| \neq 0 \end{cases} \\
(T, D_x : U_x, U_y) & \mapsto & \mathsf{utensoru}_\tau(T \textcircled{\tiny L}\, D_x, U_x, U_y) \\
\text{if } \mathsf{strategy}(T) = \textbf{right} \text{ then} & & \\
(T, U_x, V_y) & \mapsto & \begin{cases} \mathsf{uvector}_\tau(\overline{T \textcircled{\tiny R}\, V_y}) & \text{if } |T \textcircled{\tiny R}\, V_y| = 0 \\ \mathsf{umatrixu}_\tau(T \textcircled{\tiny R}\, V_y, U_x) & \text{if } |T \textcircled{\tiny R}\, V_y| \neq 0 \end{cases} \\
(T, U_x, D_y : U_y) & \mapsto & \mathsf{utensoru}_\tau(T \textcircled{\tiny R}\, D_y, U_x, U_y)
\end{array}
$$

In words, try to find a digit matrix D that can be emitted in state τ and continue with the remainder. Otherwise, more information must be absorbed from E_x or E_y according to the value of $\mathsf{strategy}(T)$. If $\mathsf{strategy}(T) = \textbf{right}$ then absorb from E_y otherwise absorb from E_x.

10 Flatten Algorithm

The basic set of functions above have to be used with care particularly where recursion is involved.

For example, consider $\sqrt{2}$. If we define the $\mathsf{sqrt2}$ recursively by

$$\mathsf{sqrt2} = \mathsf{smatrixs}_0\left(\begin{pmatrix} 1 & 2 \\ 1 & 1 \end{pmatrix}, \mathsf{sqrt2}\right)$$

then it will never output anything and never terminate.

If however, we define the $\mathsf{sqrt2}$ recursively by

$$\mathsf{sqrt2} = \mathsf{smatrixu}_0(S_+, x) \text{ where } x = \begin{pmatrix} 1 & 2 \\ 1 & 1 \end{pmatrix} : x$$

then it evaluates the answer as expected.

In general, given a mathematical formula we must be able to construct a signed expression tree and evaluate it in the following way using the function $\mathsf{sflatten}$.

$$\begin{array}{rcl}
\mathsf{sflatten} : \mathrm{sexp} & \to & \mathrm{snp}_0 \\
V & \mapsto & \mathsf{svector}_0(V) \\
M(E_x) & \mapsto & \mathsf{smatrixu}_0(M, \mathsf{uflatten}(E_x)) \\
T(E_x, E_y) & \mapsto & \mathsf{stensoru}_0(T, \mathsf{uflatten}(E_x), \mathsf{uflatten}(E_y)) \\
\mathsf{uflatten} : \mathrm{uexp} & \to & \mathrm{uexp} \\
V & \mapsto & V \\
M(E_x) & \mapsto & M(\mathsf{uflatten}(E_x)) \\
T(E_x, E_y) & \mapsto & \mathsf{utensoru}_1(T, \mathsf{uflatten}(E_x), \mathsf{uflatten}(E_y)).
\end{array}$$

Note that state 1 in $\mathsf{utensoru}_1$ can be replaced by any valid non-zero state. In words, the function $\mathsf{uflatten}$ merely flattens an unsigned expression tree into a general normal product while the function $\mathsf{sflatten}$ flattens a signed expression tree into a more specific normal product. The specific normal product is controlled by the state sets $\Phi_v(\tau)$ and $\Phi_m(\tau)$ for $\tau \in \mathbb{N}$ and the state transition function $\phi(\tau, M)$ for $\tau \in \mathbb{N}$ and $M \in \mathbb{M}$.

So, for the simple example $\sqrt{2}$, we can define the expression tree $\mathsf{sqrt2}$ recursively by

$$\mathsf{sqrt2} = \begin{pmatrix} 1 & 2 \\ 1 & 1 \end{pmatrix} (\mathsf{sqrt2})$$

and then $\mathsf{sflatten}(\mathsf{sqrt2})$ will convert it to the required normal product. For the general normal product, it remains unchanged. For unbiased exact floating point in base 2, the output is not unique due to redundancy in the representation, but it may look something like

$$S_+ : D_0^2 : D_1^2 : D_{-1}^2 : D_1^2 : D_0^2 : D_{-1}^2 : D_0^2 : D_0^2 : D_0^2 : D_0^2 : D_{-1}^2 : \ldots$$

which represents the interval $\left[\frac{1199}{849}, \frac{1200}{848}\right]$ thus far. For biased exact floating point in base 2, the output is also not unique due to redundancy in the representation, but it may look something like

$$S_+ : L : D^2_1 : D^2_{-1} : D^2_1 : D^2_0 : D^2_1 : D^2_0 : D^2_1 : D^2_0 : D^2_0 : D^2_0 : \ldots$$

which represents the interval $\left[\frac{1447}{1024}, \frac{1449}{1024}\right]$ thus far.

11 Basic Arithmetic Operations

As already mentioned, we can consider the application of a 1-dimensional lft to a normal product as an arithmetic operation:

$$\begin{aligned} M_{\mathrm{neg}}(x) &= \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix}(x) = -x \\ M_{\mathrm{rec}}(x) &= \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}(x) = \frac{1}{x} \end{aligned}$$

Given a general 1-dimensional lft $M(x)$, we can simply use the function $\mathsf{smatrixs}_0(M, x)$ to evaluate the result. So, negation and reciprocal can be defined by

$$\begin{aligned} \mathsf{sneg} : \mathrm{sexp} &\rightarrow \mathrm{snp}_0 \\ x &\mapsto \mathsf{smatrixs}_0(M_{\mathrm{neg}}, x) \\ \mathsf{srec} : \mathrm{sexp} &\rightarrow \mathrm{snp}_0 \\ x &\mapsto \mathsf{smatrixs}_0(M_{\mathrm{rec}}, x) \end{aligned}$$

Gosper [7] devised algorithms for the elementary arithmetic operations on continued fractions [22] using 2-dimensional lft's. The four most basic arithmetic operations can be represented as follows:

$$\begin{aligned} T_{\mathrm{add}}(x,y) &= \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}(x,y) = x + y \\ T_{\mathrm{sub}}(x,y) &= \begin{pmatrix} 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}(x,y) = x - y \\ T_{\mathrm{mul}}(x,y) &= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}(x,y) = x \times y \\ T_{\mathrm{div}}(x,y) &= \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}(x,y) = x \div y \end{aligned}$$

So, these operations can be defined by

$$\mathsf{sadd} : \mathrm{sexp} \times \mathrm{sexp} \rightarrow \mathrm{snp}_0$$

$$
\begin{aligned}
(x, y) &\mapsto \mathsf{stensors}_0(T_{\mathrm{add}}, x, y) \\
\mathsf{ssub} : \mathrm{sexp} \times \mathrm{sexp} &\to \mathrm{snp}_0 \\
(x, y) &\mapsto \mathsf{stensors}_0(T_{\mathrm{sub}}, x, y) \\
\mathsf{smul} : \mathrm{sexp} \times \mathrm{sexp} &\to \mathrm{snp}_0 \\
(x, y) &\mapsto \mathsf{stensors}_0(T_{\mathrm{mul}}, x, y) \\
\mathsf{sdiv} : \mathrm{sexp} \times \mathrm{sexp} &\to \mathrm{snp}_0 \\
(x, y) &\mapsto \mathsf{stensors}_0(T_{\mathrm{div}}, x, y)
\end{aligned}
$$

12 Continued Fractions

The development

$$
a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cdots}}} \tag{13}
$$

is called a *simple continued fraction* [3, 11].

The quantity

$$
r_n = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cdots + \cfrac{1}{a_n}}} \tag{14}
$$

is called the n^{th} *approximant*. The 0^{th} approximant is a_0. If the sequence r_n converges to a real number r then the continued fraction is said to be convergent and represent the number r.

Using the lft's

$$
M_n(x) = \begin{pmatrix} a_n & 1 \\ 1 & 0 \end{pmatrix}(x) = a_n + \frac{1}{x} \tag{15}
$$

we can generate the continued fraction in equation (13). Therefore, a continued fraction with non-negative coefficients for $n \geq 1$ corresponds to the general normal product

$$
M_0 : M_1 : M_2 : M_3 : \ldots
$$

Observe that $(M_0 \cdot M_1 \cdot M_2 \cdot \ldots \cdot M_n)([1, -1]) = (N_0 \cdot N_1 \cdot N_2 \cdot \ldots \cdot N_n)([0, \infty])$ where

$$
\begin{aligned}
N_{n=0} &= M_0 \cdot S_\infty = \begin{pmatrix} a_0 - 1 & a_0 + 1 \\ 1 & 1 \end{pmatrix} \\
N_{n\geq 1} &= S_\infty^{-1} \cdot M_n \cdot S_\infty = \begin{pmatrix} a_n - 2 & a_n \\ a_n & a_n + 2 \end{pmatrix}.
\end{aligned}
$$

We can see this more clearly in the following commutative diagram:

$$\begin{tikzcd}
{[1,-1]} & {[1,-1]} \arrow[l, "M_0"'] & {[1,-1]} \arrow[l, "M_1"'] & {[1,-1]} \arrow[l, "M_2"'] & \cdots \\
 & {[0,\infty]} \arrow[ul, "N_0"] \arrow[u, "S_\infty"] & {[0,\infty]} \arrow[l, "N_1"'] \arrow[u, "S_\infty"] & {[0,\infty]} \arrow[l, "N_2"'] \arrow[u, "S_\infty"] & \cdots
\end{tikzcd}$$

Therefore, a continued fraction with $|a_n| \geq 2$ for $n \geq 1$ corresponds to the general normal product

$$N_0 : N_1 : N_2 : N_3 : \ldots$$

For example, a known formula for $\tan(x)$ equates to

$$\begin{aligned}
a_{n=0} &= 0 \\
a_{n\geq 1} &= (-1)^{n+1}\frac{2n-1}{x}.
\end{aligned}$$

This leads to

$$\begin{aligned}
N_{n=0} &= \begin{pmatrix} -1 & 1 \\ 1 & 1 \end{pmatrix} \\
N_{n\geq 1} &= \begin{pmatrix} 2n-1+(-1)^n 2x & 2n-1 \\ 2n-1 & 2n-1-(-1)^n 2x \end{pmatrix}.
\end{aligned}$$

Therefore, we can derive the following expression tree for $\mathsf{tan}(S_0 : x)$

$$\begin{aligned}
\mathsf{tan} \ : \ \ \mathrm{sexp} &\to \mathrm{sexp} \\
S_0 : x &\mapsto \begin{pmatrix} 1 & 1 & -1 & -1 \\ 0 & 2 & 2 & 0 \end{pmatrix}(x, \mathsf{iterate}(\mathsf{iterator}, 0, x))
\end{aligned}$$

$$\begin{aligned}
\mathsf{iterate} \ : \ \ (\mathbb{N} \to \mathbb{T}^+) \times \mathbb{N} \times \mathrm{uexp} &\to \mathrm{uexp} \\
(T, n, x) &\mapsto T(n)(x, \mathsf{iterate}(T, n+1, x))
\end{aligned}$$

$$\begin{aligned}
\mathsf{iterator} \ : \ \ \mathbb{N} &\to \mathbb{T}^+ \\
n &\mapsto \begin{cases} \begin{pmatrix} 2n+5 & 2n+3 & 2n+1 & 2n+3 \\ 2n+3 & 2n+1 & 2n+3 & 2n+5 \end{pmatrix} & \text{if } n \text{ even} \\ \begin{pmatrix} 2n+1 & 2n+3 & 2n+5 & 2n+3 \\ 2n+3 & 2n+5 & 2n+3 & 2n+1 \end{pmatrix} & \text{if } n \text{ odd} \end{cases}
\end{aligned}$$

Observe how the recursion has been defined in order to take advantage of the preference for left absorption in the tensor absorption strategies.

For example, consider $\mathsf{tan}(\mathsf{tan}(1/3))$. Represent $1/3$ by $\begin{pmatrix} 1 \\ 3 \end{pmatrix}$. We first emit a sign matrix such as $S_0 : \begin{pmatrix} 2 \\ 1 \end{pmatrix}$. Therefore $\mathsf{tan}(1/3)$ reduces to

$$\begin{pmatrix} 1 & 1 & -1 & -1 \\ 0 & 2 & 2 & 0 \end{pmatrix}\left(\begin{pmatrix} 2 \\ 1 \end{pmatrix}, \mathsf{iterate}\left(\mathsf{iterator}, 0, \begin{pmatrix} 2 \\ 1 \end{pmatrix}\right)\right).$$

This leads to

$$\begin{pmatrix} 1 & 1 \\ 2 & 4 \end{pmatrix} \left(\mathsf{iterate} \left(\mathsf{iterator}, 0, \begin{pmatrix} 2 \\ 1 \end{pmatrix} \right) \right).$$

via left absorption because $\mathsf{strategy}\left(\left(\begin{matrix} 1 & 1 & -1 & -1 \\ 0 & 2 & 2 & 0 \end{matrix} \right) \right) = \textbf{either}$ and left absorption is the preferred direction. Again we need a sign matrix and so this reduces to

$$S_0 : \begin{pmatrix} 3 & 5 \\ 1 & 3 \end{pmatrix} \left(\mathsf{iterate} \left(\mathsf{iterator}, 0, \begin{pmatrix} 2 \\ 1 \end{pmatrix} \right) \right).$$

Hence $\mathsf{tan}(\mathsf{tan}(1/3))$ reduces to the signed expression tree

$$\begin{pmatrix} 1 & 1 & -1 & -1 \\ 0 & 2 & 2 & 0 \end{pmatrix} (x, \mathsf{iterate}(\mathsf{iterator}, 0, x))$$

where

$$x = \begin{pmatrix} 3 & 5 \\ 1 & 3 \end{pmatrix} \left(\mathsf{iterate} \left(\mathsf{iterator}, 0, \begin{pmatrix} 2 \\ 1 \end{pmatrix} \right) \right).$$

Finally, we apply the function $\mathsf{sflatten}$ for biased exact floating point in base 2 to this signed expression tree. The output is not unique due to redundancy in the representation, but it may look something like

$$S_+ : L_+ : D^2_{-1} : D^2_{-1} : D^2_1 : D^2_0 : D^2_0 : D^2_{-1} : D^2_0 : D^2_0 : D^2_1 : D^2_{-1} : \ldots$$

which represents the interval $\left[\frac{369}{1024}, \frac{370}{1024}\right]$ thus far.

A survey of various other mathematical functions in the form of general normal products has been made by Potts [20].

13 Conclusion

In conclusion, may we draw attention to the novel ideas expressed in this document.

- The use of vectors to represent rational number finitely in this setting.
- The special role of $[0, \infty]$ in representing information in an lft.
- The special role of the refinement property leading to the notion of a normal product and an extremely efficient test for interval inclusion.
- The outcome minimisation and information overlap tensor absorption strategies.
- Arbitrary base unbiased exact floating point; provides an especially efficient way to control the flow of information and the size of the integer coefficients. Roughly speaking the size of the integer coefficients increases linearly with the number of emitted digits.

- Arbitrary base biased exact floating point; creates the opportunity to add and subtract two real numbers by the manipulation of symbols and a bounded state variable.
- The flatten algorithm; this brings together many of the above ideas into a complete algorithm.
- Transformation of continued fractions into normal products.

References

[1] Algirdas Avizienis. Signed-digit number representations for fast parallel arithmetic. *IRE Transactions on electronic computers*, (10):389–400, 1961.

[2] H.J. Boehm and R. Cartwright. Exact real arithmetic: Formulating real numbers as functions. In D. Turner, editor, *Research Topics in Functional Programming*, pages 43–64. Addison-Wesley, 1990.

[3] Claude Brezinski. *History of Continued Fractions and Padé Approximants*, volume 12 of *Springer series in Computational mathematics*. Springer-Verlag, 1991.

[4] M. H. Escardó. PCF extended with real numbers. *Theoretical Computer Science*, 162(1):79–115, August 1996.

[5] Pietro Di Gianantonio. Real Number Computability and Domain Theory. In *Proceedings of the* 18^{th} *International Symposium on Mathematical Foundations of Computer Science*, pages 413–422, Gdansk, Poland, September 1993. LNCS 711.

[6] Pietro Di Gianantonio. A Golden Ratio Notation for the Real Numbers. Technical report, CWI Amsterdam, 1996.

[7] R. W. Gosper. Continued Fraction Arithmetic. Technical Report HAKMEM Item 101B, MIT AI MEMO 239, MIT, February 1972. Available from `ftp://ftp.netcom.com/pub/hb/hbaker/hakmem`.

[8] A. Grzegorczyk. On the definition of computable real continuous functions. *Fund. Math.*, 44:61–77, 1957.

[9] IEEE. IEEE Standard 754 for Binary Floating-Point Arithmetic. *SIGPLAN*, 22(2):9–25, 1985.

[10] Simon L. Peyton Jones. Arbitrary precision arithmetic using continued fractions, 1984. INDRA Note 1530, University College London.

[11] D. E. Knuth. *The Art of Computer Programming: Seminumerical Algorithms*, volume 2. Addison-Wesley, 1981.

[12] Peter Kornerup and David W. Matula. An On-line Arithmetic Unit for Bit-Pipelined Rational Arithmetic. *Journal of Parallel and Distributed Computing*, 5(3):310–330, 1988.

[13] Peter Kornerup and David W. Matula. Exploiting Redundancy in Bit-Pipelined Rational Arithmetic. In *Proceedings of the* 9^{th} *IEEE Symposium on Computer Arithmetic*, pages 119–126, Santa Monica, 1989. IEEE Computer Society Press.

[14] Peter Kornerup and David W. Matula. An Algorithm for Redundant Binary Bit-Pipelined Rational Arithmetic. *IEEE Transactions on Computers*, C-39(8):1106–1115, 1990.

[15] Peter Kornerup and David W. Matula. Finite Precision Lexicographic Continued Fraction Number Systems. In *Proceedings of the* 7^{th} *IEEE Symposium on Computer Arithmetic*, pages 207–214, Urbana, 1995. IEEE Computer Society Press.

[16] V. Menissier-Morain. Arbitrary precision real arithmetic: design and algorithms. submitted to J. Symbolic Computation, 1996.

[17] R.E. Moore. *Interval Analysis*. Prentice-Hall, Englewood Cliffs, 1966.

[18] Asger Munk Nielsen and Peter Kornerup. MSB-First Digit Serial Arithmetic. *Journal of Universal Computer Science*, 1(7):527–547, July 1995.

[19] W. Parry. On the β-expansions of real numbers. *Acta Mathematica, Acad. Sci. Hung.*, 11:401–416, 1960.

[20] P. J. Potts. Computable Real Arithmetic using Linear Fractional Transformations, June 1996. Early draft PhD Thesis, Imperial College, available from `http://www-tfm.doc.ic.ac.uk/~pjp`.

[21] J. Vuillemin. Exact real computer arithmetic with continued fractions. *IEEE Transactions on computers*, 39(8):1087–1105, August 1990.

[22] H. S. Wall. *Analytic Theory of Continued Fractions*. Chelsea Publishing Company, 1948.

[23] Osaaki Watanuki and Milos D. Ercegovac. Error Analysis of Certain Floating-Point On-Line Algorithms. *IEEE Transactions on Computers*, C-32(4):352–358, April 1983.