Semantics of Exact Real Arithmetic

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

Abstract

In this paper, we incorporate a representation of the non-negative extended real numbers based on the composition of linear fractional transformations with non-negative integer coefficients into the Programming Language for Computable Functions (PCF) with products. We present two models for the extended language and show that they are computationally adequate with respect to the operational semantics.

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 consists of 32 binary digits [11].

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.

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 that 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 [27], Boehm an Cartwright [2], Nielsen et al [20], Menissier-Morain [18], Di Gianantonio [7] and Escardó [4]. The last two authors studied extensions of PCF with a real number data type.

(ii) *Continued fraction expansions* proposed by Gosper [8], developed by Peyton Jones [12] and Vuillemin [25] and advanced more recently by Kornerup et al [17, 15, 14, 16].

(iii) *Infinite composition of linear fractional transformations* (also known as homographies or Möbius transformations) generalises the other two frameworks as demonstrated by Vuillemin [25]. Nielsen et al [20] showed that this framework can be used to represent quasi-normalised floating point [27].

Potts et al [24] have developed a framework for exact real arithmetic in which the extended real numbers are represented by the composition of linear fractional transformations with *either all non-negative or all non-positive integer coefficients*. The advantage of linear fractional transformations over linear maps is that the numerous elegant continued fractions for various mathematical functions can be used almost directly [22, 23].

In this paper, we incorporate this representation into the Programming Language for Computable Functions (PCF) with products described by Gunter [10]. This includes adding a new ground type for non-negative extended real numbers and an associated *transformation* term constructor to the grammar and also an operational semantics for program evaluation. This means that a program induced by a mathematically proved algorithm always produces the correct result.

We then present two models for the extended language and show that they are computationally adequate with respect to the operational semantics.

This paper confirms the domain-theoretic approach also used by Di Gianantonio [7] and Escardó [4] for the semantics of real number computation.

In sections 2, 3 and 4 we review the basic framework for exact real arithmetic by Potts et al [22, 23, 24]. In section 5 we present the basic notions of domain theory that we need in this paper. In section 6 we introduce the Language for Positive Reals, in section 7 we present the denotational model and finally in section 8 we prove the computational adequacy of the language.

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 [9, 19]:

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

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 ∞. For example, the closed interval $[1, -1]$ is the complement of the open interval $(-1, 1)$.

Let us make the following convenient definitions:

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

Here $\begin{vmatrix} a & c \\ b & d \end{vmatrix}$ refers to the determinant of the matrix $\begin{pmatrix} a & c \\ b & d \end{pmatrix}$.

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 $\begin{pmatrix} a & c & e & g \\ b & d & f & h \end{pmatrix} \in \mathbb{T}$. For convenience, we will use the same notation for the maximal extension of these functions.

Observe that a 1-dimensional lft of the form

$$\begin{pmatrix} a & c \\ 0 & d \end{pmatrix}(x) = \frac{a}{d} + \frac{c}{d}x$$

is a linear map. Hence, 1-dimensional lft's are generalisations of linear maps.

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

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

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. In general, we will use the letters V to denote a vector, M and N to denote matrices and T to denote 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),

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

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

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

Observe that 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

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

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 but we will ignore these since they are equivalent).*

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

Observe 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.

Note that for any rational interval $\left[\frac{a}{b}, \frac{c}{d}\right]$ contained in $[0, \infty]$, we can find a matrix M such that $\mathsf{Info}(M) = \left[\frac{a}{b}, \frac{c}{d}\right]$. For example, $M = \begin{pmatrix} a & c \\ b & d \end{pmatrix}$ or $M = \begin{pmatrix} c & a \\ d & b \end{pmatrix}$.

Proposition 2.4 *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 non-negative extended 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 satisfying the refinement property. We can denote this real number by an infinite product of matrices

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

where $M_n = \begin{pmatrix} a_n & c_n \\ b_n & d_n \end{pmatrix} \in \mathbb{M}^+$. We will call this an *infinite normal product*. This notion generalises the concept of *interval expansion* [4] in which M_n is restricted to a linear map. 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. 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.

As mentioned above, an arbitrary lft is only unique up to scaling. Hence, we can identify an lft with the equivalence classes arising from the equivalence relation $\equiv$ induced by scaling. Let us denote by P^*, the lft P reduced to its lowest terms after division by the greatest common divisor of the coefficients. We can then identify a *unique lft P^** in each equivalence class.

$$P \equiv P^* \quad (5)$$

This gives a simple representation and a convenient operational semantics for the lazy representation of the reals: finite segments of the matrix product in Equation (4) give incremental approximations to the real number in question. In particular, the first matrix tells us that the result is contained in the interval $\left[\frac{a_0}{b_0}, \frac{c_0}{d_0}\right]$ or $\left[\frac{c_0}{d_0}, \frac{a_0}{b_0}\right]$ depending on the sign of the determinant of the matrix.

3. Arithmetic Operations

Gosper [8] devised algorithms for the basic arithmetic operations on continued fractions [26] using 2-dimensional lft's. The three most basic arithmetic operations closed on $[0, \infty]$ can be represented as follows:

$$\begin{aligned} \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}(x, y) &= x + y \\ \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}(x, y) &= x \times y \\ \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}(x, y) &= x \div y \end{aligned}$$

Therefore, we need to be able to convert expressions containing vectors, matrices and tensors into normal products. In other words, we need to consider the input of normal products into a tensor, which we shall call *tensor absorption*, and the output of a normal product from a tensor, which we shall call *tensor emission*.

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:

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

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 3.1 *The following* matrix absorption *equations hold:*

$$\begin{aligned}
M(V) &= M \cdot V \\
M(N(x)) &= (M \cdot N)(x)
\end{aligned}$$

The following tensor absorption *equations hold:*

$$\begin{aligned}
T(V, y) &= \begin{cases} \overline{T \,Ⓛ\, V} & \text{if } |T \,Ⓛ\, V| = 0 \\ (T \,Ⓛ\, V)(y) & \text{if } |T \,Ⓛ\, V| \neq 0 \end{cases} \\
T(M(x), y) &= (T \,Ⓛ\, M)(x, y) \\
T(x, V) &= \begin{cases} \overline{T \circledR V} & \text{if } |T \circledR V| = 0 \\ (T \circledR V)(x) & \text{if } |T \circledR V| \neq 0 \end{cases} \\
T(x, M(y)) &= (T \circledR M)(x, y).
\end{aligned}$$

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 its mediant.

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 [24].

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

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

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:

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

The information in a 2-dimensional lft T can be represented by a 1-dimensional lft denoted by T^{head}:

$$\begin{aligned}
T^{\text{head}} &= [\sup E, \inf E] \\
E &= \{T_{ij} : i, j = 0, 1 \text{ and } T_{ij} \neq (0, 0)\} \\
\begin{pmatrix} a \\ b \end{pmatrix} \leq \begin{pmatrix} c \\ d \end{pmatrix} \quad &\text{iff} \quad \begin{vmatrix} a & c \\ b & d \end{vmatrix} \leq 0.
\end{aligned}$$

It can be shown that $\mathsf{Info}(T) = \mathsf{Info}(T^{\text{head}})$. Let us define $T^{\text{tail}} = (T^{\text{head}})^{-1} \cdot T$, where matrix inversion is defined by

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

We choose to scale the matrix inversion by the determinant in order to ensure we only get non-negative integers in the tensor emission equation:

Proposition 3.2 *The following tensor emission equation holds:*

$$T(x,y) = T^{\text{head}}(T^{\text{tail}}(x,y)).$$

This corresponds to the extraction of maximum information known as *naive emission*. There are alternative emission methods that are considerably more efficient [22, 24].

4. Continued Fractions

The development

$$a_0 + \cfrac{b_0}{a_1 + \cfrac{b_1}{a_2 + \cfrac{b_2}{a_3 + \cdots}}} \tag{6}$$

is called a *continued fraction* [3, 13].

The quantity

$$r_n = a_0 + \cfrac{b_0}{a_1 + \cfrac{b_1}{a_2 + \cdots + \cfrac{b_{n-1}}{a_n}}} \tag{7}$$

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

$$M_n(x) = \begin{pmatrix} a_n & b_n \\ 1 & 0 \end{pmatrix}(x) = a_n + \frac{b_n}{x} \tag{8}$$

we can generate the continued fraction in Equation (6). Therefore, a continued fraction with non-negative coefficients corresponds to a normal product.

A survey of various mathematical functions in the form of general normal products has been made by Potts [22].

5. Domain Theory

We have a choice between a continuous domain and an algebraic domain for the real numbers.

For the continuous domain, it is convenient to expand on the usual notation. Let x and y be elements of a dcpo D. We say that x is *way-below* or *approximates* y, denoted $x \ll y$, if for all directed subsets A of D, $y \sqsubseteq \bigsqcup A$ implies $\exists a \in A$ such that $x \sqsubseteq a$. We say that x is *compact* if it approximates itself. It can be shown that if $x \ll y$ then $\exists z \in D$ such that $x \ll z$ and $z \ll y$.

Let us define $\uparrow x = \{y \in D \mid x \sqsubseteq y\}$, $\downarrow x = \{y \in D \mid y \sqsubseteq x\}$, $\twoheaduparrow x = \{y \in D \mid x \ll y\}$, $\twoheaddownarrow x = \{y \in D \mid y \ll x\}$ and $K(D) = \{x \in D \mid x \text{ is compact}\}$.

We say that a subset B of a dcpo D is a *basis* for D, if for every element x of D the set $\twoheaddownarrow x \cap B$ contains a directed subset with supremum x.

A dcpo is called a *continuous domain* if it has a basis.

A dcpo is called an *algebraic domain* if it has a basis of compact elements.

Given a transitive relation $\prec$ on a set B, known as an *abstract basis*, $A \subseteq B$ is an *ideal* if it is downward closed ($A = \{b \in B \mid \forall a \in A. b \prec a\}$) and directed ($\forall a, b \in A. \exists c \in A$ such that $a \prec c$ and $b \prec c$). The set of ideals is denoted $\overline{(B, \prec)}$. The *ideal completion* of $(B, \prec)$ is the set of ideals ordered by set inclusion.

5.1. A Continuous Domain for the Real Numbers

The closed intervals of $[0,\infty]$ ordered by reverse inclusion is a continuous domain $\mathrm{I}^C[0,\infty]$ with a basis given by the closed rational intervals $\mathrm{I}\mathbb{Q}^+$ where $\mathbb{Q}^+ = [0,\infty] \cap \mathbb{Q}$ including infinity. The ideal completion of the abstract basis $(\mathrm{I}\mathbb{Q}^+, \ll)$ is isomorphic to the continuous domain $\mathrm{I}^C[0,\infty]$. The maximal elements (singleton sets) are identified with the non-negative extended real numbers.

For a continuous function $f : \mathrm{I}^C[0,\infty] \to \mathrm{I}^C[0,\infty]$, if $y \ll f(x)$ with $y \in \mathrm{I}\mathbb{Q}^+$ then $\exists z \ll x$ such that $y \ll f(z)$ with $z \in \mathrm{I}\mathbb{Q}^+$.

5.2. An Algebraic Domain for the Real Numbers

The ideal completion of $(\mathrm{I}\mathbb{Q}^+, \supseteq)$ is an algebraic domain $\mathrm{I}^A[0,\infty]$ with basis isomorphic to $\mathrm{I}\mathbb{Q}^+$.

This domain has the advantage of allowing a distinction to be made between a finite and infinitely represented rational number [7]. This is useful for efficiency reasons as we do not want to be forced to represent a rational number by an infinite product of matrices if it can be avoided.

Given an interval $x \in \mathrm{I}^C[0,\infty]$, let us use the notation $\underline{x} = \inf x$, $\overline{x} = \sup x$ and

$$\begin{aligned}
\langle x \rangle &= \{y \in \mathrm{I}\mathbb{Q}^+ \mid \underline{y} \le \underline{x} \text{ and } \overline{x} \le \overline{y}\} \\
\langle x \rangle\rangle &= \{y \in \mathrm{I}\mathbb{Q}^+ \mid \underline{y} \le \underline{x} \text{ and } \overline{x} < \overline{y}\} \\
\langle\langle x \rangle &= \{y \in \mathrm{I}\mathbb{Q}^+ \mid \underline{y} < \underline{x} \text{ and } \overline{x} \le \overline{y}\} \\
\langle\langle x \rangle\rangle &= \{y \in \mathrm{I}\mathbb{Q}^+ \mid \underline{y} < \underline{x} \text{ and } \overline{x} < \overline{y}\}
\end{aligned}$$

where $0 < 0$ and $\infty < \infty$. Note that $\langle x \rangle \equiv \downarrow x$ and $\langle\langle x \rangle\rangle \equiv \twoheaddownarrow x$.

In the algebraic domain, the elements $\langle x \rangle$, $\langle x \rangle\rangle$, $\langle\langle x \rangle$ and $\langle\langle x \rangle\rangle$ are distinct whenever $x \in \mathrm{I}\mathbb{Q}^+$, they provide only two distinct elements whenever $x \in \mathrm{I}^C[0,\infty]$ with one rational end-point and they are indistinguishable whenever $x \in \mathrm{I}^C[0,\infty]$ with irrational end-points.

Note that the functions $e : \mathrm{I}^C[0,\infty] \to \mathrm{I}^A[0,\infty]$ with $x \mapsto \langle\langle x\rangle\rangle$ and $p : \mathrm{I}^A[0,\infty] \to \mathrm{I}^C[0,\infty]$ with $x \mapsto \bigcap x$ form an embedding/projection pair with $p \circ e = \mathrm{Id}$ and $e \circ p \sqsubseteq \mathrm{Id}$.

For a continuous function $f : \mathrm{I}^A[0,\infty] \to \mathrm{I}^A[0,\infty]$, if $y \sqsubseteq f(x)$ with y compact then $\exists z \sqsubseteq x$ such that $y \sqsubseteq f(z)$ with z compact.

6. Language for Positive Reals

The Language for Positive Reals (LPR) includes the syntax and conventions of the Programming Language for Computable Functions (PCF) with products described by Gunter [10]. This in turn includes the terms of the simply-typed λ-calculus.

The context-free grammar for LPR is given in BNF by

$$
\begin{array}{rcl}
x & \in & \text{Variable} \\
t & ::= & \mathbf{num} \mid \mathbf{bool} \mid \mathbf{I} \mid t \to t \mid t \times t \\
P & ::= & x \mid \mathbf{0} \mid \mathbf{true} \mid \mathbf{false} \mid \\
 & & \mathbf{succ}(P) \mid \mathbf{pred}(P) \mid \mathbf{zero?}(P) \mid \\
 & & \mathbf{if}\ P\ \mathbf{then}\ P\ \mathbf{else}\ P \mid \\
 & & \lambda x : t.P \mid PP \mid \mu x : t.P \mid \\
 & & (P,P) \mid \mathbf{fst}(P) \mid \mathbf{snd}(P) \mid \langle P\rangle
\end{array}
$$

where Variable is the primitive syntax class of *variables*. The expressions in the syntax class over which t ranges are called *types*, and those over which P ranges are called *term trees*.

The types **num**, **bool** and **I** are called the *ground* types.

Term trees of the form $\lambda x : t.P$ are called *abstractions*, and those of the form PQ are called *applications*. The other constructs of PCF with products include the successor – $\mathbf{succ}(P)$, predecessor – $\mathbf{pred}(P)$, test for zero – $\mathbf{zero?}(P)$, conditional – $\mathbf{if}\ P\ \mathbf{then}\ Q\ \mathbf{else}\ R$, pairing – (P,Q), first projection – $\mathbf{fst}(P)$, second projection – $\mathbf{snd}(P)$ and recursion – $\mu x : t.P$. The new construct for LPR is *transformation* – $\langle P\rangle$.

The equivalence class of term trees modulo renaming of bound variables are called just *terms* and we refer to closed terms of ground type as *programs*. We will use the notation $[Q/x]P$ for *substitution* to indicate the result of replacing all free occurrences of the variable x in P by Q, making the appropriate changes in the bound variables of P so that no free variables in Q become bound.

For convenience, $\mathbf{vector}(t)$ denotes $t \times t$, $\mathbf{matrix}(t)$ denotes $\mathbf{vector}(t) \times \mathbf{vector}(t)$ and $\mathbf{tensor}(t)$ denotes $\mathbf{matrix}(t) \times \mathbf{matrix}(t)$.

There are two systems of rules describing LPR.

The first of these determines which of the term described by the syntax above are to be considered *well-typed*. These are the terms to which we will assign a meaning in our semantic model.

The second set of rules form the *operational semantics* for evaluation.

6.1. Typing Rules

A *type assignment* is a list $H \equiv x_1 : t_1, x_2 : t_2, \ldots, x_n : t_n$ of pairs of variables and types such that the variables are distinct.

A *typing judgement* is a triple, denoted $H \vdash P : t$, consisting of a type assignment H, a term P and a type t such that all the free variables of P appear in the list H. We read this triple as "given the assignment H, the term P has type t". It is defined to be the least relation satisfying the typing rules for PCF with products, which is well known [10], and those below:

$$
\begin{array}{ll}
[\text{VecNum}] & \dfrac{H \vdash V : \mathbf{vector(num)}}{H \vdash \langle V\rangle : \mathbf{I}} \\[2ex]
[\text{MatNum}] & \dfrac{H \vdash M : \mathbf{matrix(num)}}{H \vdash \langle M\rangle : \mathbf{I} \to \mathbf{I}} \\[2ex]
[\text{TenNum}] & \dfrac{H \vdash T : \mathbf{tensor(num)}}{H \vdash \langle T\rangle : \mathbf{I} \times \mathbf{I} \to \mathbf{I}}
\end{array}
$$

6.2. Reduction Rules

The *strategy* for evaluating a program is called an *operational semantics* for the language. One approach to describing such a semantics is to indicate how a term P evaluates to another term Q by defining a relation $P \to Q$ between terms using a set of *one-step reduction rules*. We define the *one-step reduction relation* $\to$ to be the least relation satisfying the one-step reduction rules for *call-by-name evaluation* of PCF with products, which is well known [10], and those below:

$$
\begin{array}{lc}
\langle\text{UniqVec}\rangle & \langle V\rangle \to \langle V^*\rangle \\
\langle\text{UniqMat}\rangle & \langle M\rangle P \to \langle M^*\rangle P \\
\langle\text{UniqTen}\rangle & \langle T\rangle P \to \langle T^*\rangle P \\
\langle\text{MatAbs1}\rangle & \langle M\rangle\langle V\rangle \to \langle M \cdot V\rangle \\
\langle\text{MatAbs2}\rangle & \langle M\rangle(\langle N\rangle P) \to \langle M \cdot N\rangle P \\[1ex]
\langle\text{TenAbs1}\rangle & \dfrac{|T \textcircled{\scriptsize L} V| \neq 0}{\langle T\rangle(\langle V\rangle, P) \to \langle T \textcircled{\scriptsize L} V\rangle P} \\[2ex]
\langle\text{TenAbs2}\rangle & \dfrac{|T \textcircled{\scriptsize L} V| = 0}{\langle T\rangle(\langle V\rangle, P) \to \langle \underline{T \textcircled{\scriptsize L} V} \equiv \overline{T \textcircled{\scriptsize L} V}\rangle}
\end{array}
$$

$$
\begin{array}{ll}
\langle\text{TenAbs3}\rangle & \langle T\rangle(\langle M\rangle P, Q) \to \langle T \textcircled{L}\, M\rangle(P,Q) \\[6pt]
\langle\text{TenAbs4}\rangle & \dfrac{|T \circledR\, V| \neq 0}{\langle T\rangle(P, \langle V\rangle) \to \langle T \circledR\, V\rangle P} \\[10pt]
\langle\text{TenAbs5}\rangle & \dfrac{|T \circledR\, V| = 0}{\langle T\rangle(P, \langle V\rangle) \to \langle \underline{T \circledR\, V} \equiv \overline{T \circledR\, V}\rangle} \\[10pt]
\langle\text{TenAbs6}\rangle & \langle T\rangle(P, \langle M\rangle Q) \to \langle T \circledR\, M\rangle(P,Q) \\[6pt]
\langle\text{Emission}\rangle & \dfrac{\mathsf{Info}(T) \neq \bot}{\langle T\rangle(P,Q) \to \langle T^{\text{head}}\rangle(\langle T^{\text{tail}}\rangle(P,Q))} \\[10pt]
\langle\text{Cong1}\rangle & \dfrac{V \to W}{\langle V\rangle \to \langle W\rangle} \\[10pt]
\langle\text{Cong2}\rangle & \dfrac{M \to N \quad P \to Q}{\langle M\rangle P \to \langle N\rangle Q} \\[10pt]
\langle\text{Cong3}\rangle & \dfrac{T \to U \quad P \to Q}{\langle T\rangle P \to \langle U\rangle Q}
\end{array}
$$

The rules $\langle\text{UniqVec}\rangle$, $\langle\text{UniqMat}\rangle$ and $\langle\text{UniqTen}\rangle$ arise from the equivalence relation induced by scaling in Equation (5). The rules $\langle\text{MatAbs1/2}\rangle$ and $\langle\text{TenAbs1-6}\rangle$ arise from the absorption equations in Proposition 3.1. The rule $\langle\text{Emission}\rangle$ arises from the emission equation in Proposition 3.2. Finally, the transformation construct has three associated congruence rules $\langle\text{Cong1-3}\rangle$.

Note that $\mathsf{Info}(T) \neq \bot$ if one of the vectors T_{00}, T_{01}, T_{10} and T_{11} is equivalent to 0 or ∞.

The *reduction relation* $\to^*$ is defined as the reflexive, transitive closure of the one-step reduction relation.

We say that a term P evaluates to a value Z if $P \to^* Z$ where a value is a term generated by the following grammar:

$$
\begin{aligned}
Z \quad ::= \quad & \mathbf{0} \mid \mathbf{true} \mid \mathbf{false} \mid \mathbf{succ}(Z) \mid \lambda x : t.P \mid \\
& (Z,Z) \mid \left\langle \begin{array}{c} Z \\ Z \end{array} \right\rangle \mid \left\langle \begin{array}{cc} Z & Z \\ Z & Z \end{array} \right\rangle P
\end{aligned}
$$

Evaluation is a multi-valued function Eval from programs to values.

$$
\begin{aligned}
\text{Eval}(P) \quad = \quad & \{\bot\} \cup \\
& \{n \mid P \to^* \mathbf{succ}^n(0)\} \cup \\
& \{\text{true} \mid P \to^* \mathbf{true}\} \cup \\
& \{\text{false} \mid P \to^* \mathbf{false}\} \cup \\
& \{\mathsf{Info}(V) \mid P \to^* \langle V\rangle\} \cup \\
& \{\mathsf{Info}(M) \mid P \to^* \langle M\rangle Q \text{ for some Q}\}
\end{aligned}
$$

6.3. Example

The Stieltjes type continued fraction for $\arctan x$ is given [5, 6] by

$$
\arctan x = \cfrac{x}{1 + \cfrac{\frac{x^2}{3}}{1 + \cfrac{\frac{4x^2}{15}}{1 + \cdots}}}
$$

This can be transformed to

$$
\arctan x = \prod_{n=0}^{\infty} \left(\begin{array}{cc} 0 & x \\ (1+n)^2 x & 1+2n \end{array} \right)
$$

or put another way

$$
\begin{aligned}
\arctan x = & \\
& \left(\begin{array}{cccc} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{array} \right)(x, \left(\begin{array}{cccc} 0 & 1 & 0 & 0 \\ 4 & 0 & 0 & 3 \end{array} \right)(x, \\
& \left(\begin{array}{cccc} 0 & 1 & 0 & 0 \\ 9 & 0 & 0 & 5 \end{array} \right)(x, \left(\begin{array}{cccc} 0 & 1 & 0 & 0 \\ 16 & 0 & 0 & 7 \end{array} \right)(x, \cdots)))).
\end{aligned}
$$

Thus, a program for arctan in LPR is

$$
\arctan = \lambda x.\, (\mu f.\lambda n.\mathrm{I}^{\text{ARCTAN}}(n)(x, f(n+1)))\, 0
$$

where the arctan iterator $\mathrm{I}^{\text{ARCTAN}}$ is

$$
\mathrm{I}^{\text{ARCTAN}} = \lambda n. \left\langle \begin{array}{cccc} 0 & 1 & 0 & 0 \\ (1+n)^2 & 0 & 0 & 1+2n \end{array} \right\rangle.
$$

7. Models

The ground types **num**, **bool** and **I** in the language are interpreted by the flat domain of natural numbers $\mathbb{N}_\bot = \mathbb{N} \cup \{\infty\}$, the flat domain of truth values $\mathbb{T} = \{\text{true}, \text{false}, \bot\}$ and the continuous domain $\mathrm{I}^C[0,\infty]$ or the algebraic domain $\mathrm{I}^A[0,\infty]$ of intervals over the non-negative extended real numbers.

We will use *semantic brackets* $[\![]\!]$ to distinguish between terms in the language and expressions in the model.

Extending the *standard fixed-point model* relative to call-by-name of PCF with products to LPR, the interpretation $[\![t]\!]$ of a type t is a dcpo defined inductively by

$$
\begin{aligned}
[\![\mathbf{num}]\!] &= \mathbb{N}_\bot \\
[\![\mathbf{bool}]\!] &= \mathbb{T} \\
[\![\mathbf{I}]\!] &= \mathrm{I}^C[0,\infty] \text{ or } \mathrm{I}^A[0,\infty] \\
[\![s \to t]\!] &= [[\![s]\!] \to [\![t]\!]] \\
[\![s \times t]\!] &= [\![s]\!] \times [\![t]\!].
\end{aligned}
$$

While a type assignment associates *types* with variables, an *environment* associates *values* to variables. If H is a type assignment, then an *H-environment* is a function ρ on variables that maps each $x : t \in H$ to a value $\rho(x) \in [\![t]\!]$.

Let us use the notation $[\![H \rhd P : t]\!]$ for the interpretation of term P relative to type assignment H and type t. Thus $[\![H \rhd P : t]\!]$ is a function from H-environments to $[\![t]\!]$ defined by induction on the type derivation of $H \vdash P : t$. Thus, for transformation $\langle P \rangle$ we have $[\![H \rhd \langle P \rangle : t]\!]\rho = P$ in the continuous domain $\mathrm{I}^C[0,\infty]$ and $[\![H \rhd \langle P \rangle : t]\!]\rho = \downarrow P$ in the algebraic domain $\mathrm{I}^A[0,\infty]$. Recall that the P outside the semantic brackets is suppose to represent the maximal extension of the arbitrary lft represented by P.

8. Computational Adequacy

The following proposition is easily proved:

Proposition 8.1 (Soundness) *For all programs P,*

$$\bigsqcup \mathit{Eval}(P) \sqsubseteq [\![P]\!]$$

In order to establish completeness, we consider Plotkin's notion of computability [21] as extended by Escardó [4]:

Definition 8.2 *The* computable terms *of LPR form the least set of terms such that*

- *If P is a program then P is computable whenever $[\![P]\!] \sqsubseteq \bigsqcup \mathit{Eval}(P)$.*
- *If $\vdash P : s \to t$ then P is computable whenever PQ is computable for any closed computable term Q of type s.*
- *If $x_1 : t_1, x_2 : t_2, \ldots, x_n : t_n \vdash P : t$ then P is computable whenever $[P_1, P_2, \ldots, P_n / x_1, x_2, \ldots, x_n]P$ is computable for any set of closed computable terms P_i such that $\vdash P_i : t_i$.*

So, we need to prove that every term is computable, establishing Theorem 8.6 below, by extending the inductive proof of Plotkin [21] with the following lemmas. Note that the equivalent lemmas with respect to the algebraic domain $\mathrm{I}^A[0,\infty]$ can be derived by replacing all instances of "$x \ll y$" by "compact $x \sqsubseteq y$".

Lemma 8.3 *A real program P is* computable *with respect to the continuous domain $I^C[0,\infty]$ iff*

$$\forall a \ll [\![P]\!] \text{ with } a \neq \bot \Rightarrow \exists b \in \mathit{Eval}(P) \text{ with } a \sqsubseteq b$$

Lemma 8.4 *$\langle T \rangle$ is computable with respect to the continuous domain $I^C[0,\infty]$*

Proof We need to show that $\langle T \rangle PQ$ is computable if P and Q are computable. Thus, let P and Q be computable terms and let $a \ll [\![\langle T \rangle PQ]\!] = T[\![P]\!][\![Q]\!]$ with $a \neq \bot$. By continuity of T, $\exists c \ll [\![P]\!]$ and $\exists d \ll [\![Q]\!]$ such that $a \ll Tcd$.

- If $c = \bot$ and $d = \bot$ then $\langle T \rangle PQ \to \langle T^{\mathrm{head}} \rangle (\langle T^{\mathrm{tail}} \rangle PQ)$ so let $b = \mathsf{Info}((\langle T^{\mathrm{head}} \rangle))$. But $b \in \mathrm{Eval}(\langle T \rangle PQ)$ and $a \sqsubseteq b$ because $\mathsf{Info}(T) \sqsubseteq \mathsf{Info}((T^{\mathrm{head}}))$. Hence $\langle T \rangle PQ$ is a computable term.
- If $c = \bot$ and $d \neq \bot$ then $\exists f \in \mathrm{Eval}(Q)$ with $d \sqsubseteq f$ because Q is a computable term.
  - If f is a singleton then $\exists V$ such that $f = \mathsf{Info}(V)$ and $Q \to^* \langle V \rangle$. Therefore $\langle T \rangle PQ \to^* \langle T \rangle P \langle V \rangle \to \langle T \textcircled{R} V \rangle P$. Let $b = \mathsf{Info}((T \textcircled{R} V))$. But $b \in \mathrm{Eval}(\langle T \rangle PQ)$ and $a \sqsubseteq b$ because $\mathsf{Info}(T) \sqsubseteq \mathsf{Info}((T \textcircled{R} V))$. Hence $\langle T \rangle PQ$ is a computable term.
  - If f is not a singleton then $\exists M$ such that $f = \mathsf{Info}(M)$ and $Q \to^* \langle M \rangle R$ for some R. Therefore $\langle T \rangle PQ \to^* \langle T \rangle P(\langle M \rangle R) \to \langle T \textcircled{R} M \rangle PR \to \langle (T \textcircled{R} M)^{\mathrm{head}} \rangle (\langle (T \textcircled{R} M)^{\mathrm{tail}} \rangle PR)$. Let $b = \mathsf{Info}(((T \textcircled{R} M)^{\mathrm{head}}))$. But $b \in \mathrm{Eval}(\langle T \rangle PQ)$ and $a \sqsubseteq b$ because $\mathsf{Info}(T) \sqsubseteq \mathsf{Info}(((T \textcircled{R} M)^{\mathrm{head}}))$. Hence $\langle T \rangle PQ$ is a computable term.
- Similarly if $c \neq \bot$ and $d = \bot$.
- If $c \neq \bot$ and $d \neq \bot$ then $\exists e \in \mathrm{Eval}(P)$ with $c \sqsubseteq e$ and $\exists f \in \mathrm{Eval}(Q)$ with $d \sqsubseteq f$ because P and Q are computable terms respectively.
  - If e and f are singletons then $\exists V$ such that $e = \mathsf{Info}(V)$ and $P \to^* \langle V \rangle$ and $\exists W$ such that $f = \mathsf{Info}(W)$ and $Q \to^* \langle W \rangle$. Therefore $\langle T \rangle PQ \to^* \langle T \rangle \langle V \rangle \langle W \rangle \to^* \langle T \textcircled{L} V \textcircled{R} W \rangle$. Let $b = \mathsf{Info}((T \textcircled{L} V \textcircled{R} W))$. But $b \in \mathrm{Eval}(\langle T \rangle PQ)$ and $a \sqsubseteq b$ because $\mathsf{Info}(T) \sqsubseteq \mathsf{Info}((T \textcircled{L} V \textcircled{R} W))$. Hence $\langle T \rangle PQ$ is a computable term.
  - If e is a singleton, but f is not then $\exists V$ such that $e = \mathsf{Info}(V)$ and $P \to^* \langle V \rangle$ and $\exists N$ such that $f = \mathsf{Info}(N)$ and $Q \to^* \langle N \rangle S$ for some S. Therefore $\langle T \rangle PQ \to^* \langle T \rangle \langle V \rangle (\langle N \rangle S) \to^* \langle T \textcircled{L} V \textcircled{R} N \rangle S$. Let $b = \mathsf{Info}((T \textcircled{L} V \textcircled{R} N))$. But $b \in \mathrm{Eval}(\langle T \rangle PQ)$ and $a \sqsubseteq b$ because $\mathsf{Info}(T) \sqsubseteq \mathsf{Info}((T \textcircled{L} V \textcircled{R} N))$. Hence $\langle T \rangle PQ$ is a computable term.
  - Similarly if f is a singleton, but e is not.
  - If e and f are not singletons then

$\exists M$ such that $e = \mathsf{Info}(M)$ and $P \to^* \langle M \rangle R$ for some R and $\exists N$ such that $f = \mathsf{Info}(N)$ and $Q \to^* \langle N \rangle S$ for some S. Therefore $\langle T \rangle PQ \to^* \langle T \rangle (\langle M \rangle R)(\langle N \rangle S) \to^* \langle T \textcircled{L} M \textcircled{R} N \rangle RS \to^* \langle (T \textcircled{L} M \textcircled{R} N)^{\mathrm{head}} \rangle (\langle (T \textcircled{L} M \textcircled{R} N)^{\mathrm{tail}} \rangle RS)$. Let

$b = \mathsf{Info}(((T \circledL M \circledR N)^{\text{head}}))$. But $b \in \text{Eval}(\langle T\rangle PQ)$ and $a \sqsubseteq b$ because $\mathsf{Info}(T) \sqsubseteq \mathsf{Info}(((T \circledL M \circledR N)^{\text{head}}))$. Hence $\langle T\rangle PQ$ is a computable term. □

Lemma 8.5 *Every term P of LPR is computable.*

Proof The proof is by structural induction on P. Let σ be a substitution of closed computable terms for the free variables in P. Thus, we must show that σP is computable.

- For $P \equiv x$, $\mathbf{0}$, **true**, **false**, $\mathbf{succ}(Q)$, $\mathbf{pred}(Q)$, $\mathbf{zero?}(Q)$, **if** Q **then** R **else** S, (Q, R), $\mathbf{fst}(Q)$ or $\mathbf{snd}(Q)$, P is computable because it is in PCF with products.
- For $P \equiv \lambda x : t.Q$, we must show that $R \equiv (\lambda x : t.\sigma Q)P_1P_2 \ldots P_n$ is computable if $P_1, P_2, \ldots, P_n$ are closed computable terms and R has ground type s.
  - For $s \equiv \mathbf{num}$ and **bool**, R is computable because it is in PCF with products.
  - For $s \equiv \mathbf{I}$, observe that $R \rightarrow ([P_1/x](\sigma Q))P_2P_3 \ldots P_n \equiv ((\sigma[P_1/x])Q)P_2P_3 \ldots P_n$, which we will call S. But, S is computable because Q is computable. Therefore, $[\![R]\!] \sqsubseteq \bigsqcup \text{Eval}(R)$ because $[\![R]\!] = [\![S]\!]$, $\bigsqcup \text{Eval}(R) = \bigsqcup \text{Eval}(S)$ and $[\![S]\!] \sqsubseteq \bigsqcup \text{Eval}(S)$.
- For $P \equiv \mu x : t.Q$, we must show that $R \equiv SP_1P_2 \ldots P_n$ where $S \equiv \mu x : t.\sigma Q$ is computable if $P_1, P_2, \ldots, P_n$ are closed computable terms and R has ground type s.
  - For $s \equiv \mathbf{num}$ and **bool**, R is computable because it is in PCF with products.
  - For $s \equiv \mathbf{I}$, define S^n by
  $$\begin{aligned} S^0 &\equiv \mu x : t.x \\ S^{n+1} &\equiv (\lambda x : t.\sigma Q)S^n. \end{aligned}$$
  It is easy to show by induction on n that $[\![S]\!] = \bigsqcup_{n\in\mathbb{N}}[\![S^n]\!]$. Clearly, S^n is computable for all $n \in \mathbb{N}$.
  Let $a \ll [\![R]\!]$ with $a \neq \bot$.
  $$\begin{aligned} [\![R]\!] &= [\![S]\!][\![P_1]\!][\![P_2]\!] \ldots [\![P_n]\!] \\ &= (\bigsqcup_{n\in\mathbb{N}}[\![S^n]\!])[\![P_1]\!][\![P_2]\!] \ldots [\![P_n]\!] \\ &= \bigsqcup_{n\in\mathbb{N}}([\![S^n]\!][\![P_1]\!][\![P_2]\!] \ldots [\![P_n]\!]) \\ &= \bigsqcup_{n\in\mathbb{N}}[\![S^nP_1P_2 \ldots P_n]\!] \end{aligned}$$
  However, $\exists n \in \mathbb{N}$ such that $a \ll [\![S^nP_1P_2 \ldots P_n]\!]$, therefore $\exists b \in \text{Eval}(S^nP_1P_2 \ldots P_n)$ with $a \sqsubseteq b$ because S^n is computable. Therefore $b \in \text{Eval}(R)$ by a straightforward extension of the Unwinding Theorem [10].
- For $P \equiv \langle Q\rangle$, Q may be a vector, matrix or tensor. The tensor case is proved in Lemma 8.4 and the other two are just special cases. □

Theorem 8.6 (Completeness) *For all programs P,*
$$[\![P]\!] \sqsubseteq \bigsqcup Eval(P)$$

The soundness and completeness properties are together called the *computational adequacy property.*

In section 3, we defined a specific absorption strategy for a tensor $T(x, y)$ that decides whether to absorb from x or y. This fair strategy was not incorporated into the language LPR and so we cannot say anything about its correctness. However, computational adequacy does say that a correct strategy must exist and that it must be fair.

9. Conclusion

In this paper, we define a Language for Positive Reals (LPR) that incorporates a representation of the non-negative extended real numbers based on the composition of linear fractional transformations with non-negative integer coefficients into the Programming Language for Computable Functions (PCF) with products. This representation allows a wide range of mathematical functions including the basic arithmetic operations and transcendental functions to be defined elegantly because of the simple connection with the rich theory of continued fractions.

We have shown that LPR is computational adequate with respect to both the continuous domain and the algebraic domain of real numbers. This result means that a mathematical proof of correctness of a recursive algorithm is sufficient to conclude that a program induced by it produces the correct result. In other words, a syntactic proof for the program by appealing to the operational semantics is unnecessary.

This framework for exact real arithmetic in $[0, \infty]$ has been extended to $\mathbb{R}^\infty$ by prefixing normal products with an arbitrary integer coefficient lft [24]. Consider for example the natural cover of $\mathbb{R}^\infty$ by the four intervals $[0, \infty]$, $[1, -1]$, $[\infty, 0]$ and $[-1, 1]$. Four lft's map each of the above intervals to the interval $[0, \infty]$. A real number can be located in one of the above quarters in finite time. Therefore, by using one of the four lft's above, its computation can be eventually made in the interval $[0, \infty]$. The Language for Positive Reals can be extended to a language for all reals using the above framework.

Comparison of real numbers may be implemented using the quasi-relational comparison operator $<_\epsilon$ described by Boehm and Cartwright [2].

Finally, we note that this framework could also be extended to cater for interval inputs and interval outputs.

References

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

[2] H. 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] C. 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 Comput. Sci.*, 162(1):79–115, August 1996.

[5] L. Euler. Commentatio in fractionem continuam qua illustris La Grange potestates binomiales expressit. *Mémoires Acad. impér. Sci. Petersb.*, 6:3–11, 1813–1814.

[6] L. Euler. An Essay on Continued Fractions. *Maths. Systems Theory*, 18:295–328, 1985.

[7] P. D. 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.

[8] 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`.

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

[10] C. A. Gunter. *Semantics of Programming Languages – Structures and Techniques*. The MIT Press, 1992.

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

[12] S. L. P. Jones. Arbitrary precision arithmetic using continued fractions, 1984. INDRA Note 1530, University College London.

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

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

[15] P. Kornerup and D. 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.

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

[17] P. Kornerup and D. 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.

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

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

[20] A. M. Nielsen and P. Kornerup. MSB-First Digit Serial Arithmetic. *Journal of Universal Computer Science*, 1(7):527–547, July 1995.

[21] G. D. Plotkin. Post-graduate lecture notes in advanced domain theory (incorporating the "Pisa Notes"). Dept. of Computer Science, Univ. of Edinburgh. Available in LaTeX, 1981.

[22] 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`.

[23] P. J. Potts and A. Edalat. Exact Real Arithmetic based on Linear Fractional Transformations, December 1996. EPSRC project, Imperial College, available from `http://www-tfm.doc.ic.ac.uk/~pjp`.

[24] P. J. Potts and A. Edalat. Exact Real Computer Arithmetic, March 1997. EPSRC project, Imperial College, available from `http://www-tfm.doc.ic.ac.uk/~pjp`.

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

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

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