Lazy Computation with Exact Real Numbers*

Abbas Edalat   Peter John Potts   Philipp Sünderhauf

Department of Computing, Imperial College, 180 Queen's Gate, London SW7 2BZ, UK
{a.edalat,p.potts,p.sunderhauf}@doc.ic.ac.uk

Abstract

We provide a semantical framework for exact real arithmetic using linear fractional transformations on the extended real line. We present an extension of PCF with a real type which introduces an eventually breadth-first strategy for lazy evaluation of exact real numbers. In this language, we present the constant *redundant if*, rif, for defining functions by cases which, in contrast to *parallel if* (pif), overcomes the problem of undecidability of comparison of real numbers in finite time. We use the upper space of the one-point compactification of the real line to develop a denotational semantics for the lazy evaluation of real programs. Finally two adequacy results are proved, one for programs containing rif and one for those not containing it. Our adequacy results in particular provide the proof of correctness of algorithms for computation of single-valued elementary functions.

1 Introduction

It is well known that the accumulation of round-off errors in floating point programs can lead to grossly incorrect numerical results which in general are impossible to estimate [2]. This problem, with all its implications, has remained the main unresolved issue in computer arithmetic.

Two main approaches have been proposed for exact real arithmetic as an alternative to floating point: The first, introduced by Vuillemin [20], uses redundant continued fractions to represent real numbers and has been implemented by Lester [10]. The other approach, by Boehm and Cartwright, uses redundant sequences of B-adic numbers to represent reals and has been implemented by Menissier-Morain [11]. There has been no attempt to provide proper semantics for either of the two frameworks. In fact, there are no proofs of correctness of algorithms in the former and the lack of semantics in the latter makes it very hard to check the various ad hoc proofs for the correctness of the algorithms given in [11].

A new framework for exact real computation has been introduced in [15, 6] in which a real number is given by the intersection of a nested sequence of rational intervals obtained by applying a sequence of composition of linear fractional transformations (lft's), equivalently matrices, with integer coefficients to the base interval $[0, \infty]$. This provides a redundant representation of a real number by a product of matrices, called a *normal product*. A complete set of efficient on-line algorithms for elementary functions in this setting has been provided in [13] and implemented in Caml. It has the distinguishing features that it is incremental (i.e. to obtain a more accurate result one only needs to proceed with computation at the previous stage) and also includes rational arithmetic.

In [16] following Scott's suggestion in [17] and the work in [4, 7], an extension of PCF [18, 12], with a real number data type, based on the domain of intervals, was presented to give a denotational semantics to this lft framework. However, there are two essential restrictions here: Firstly, it only applies to computation on the non-negative reals which greatly simplifies the framework as one only needs to work with matrices with non-negative integer coefficients. Secondly, it uses non-deterministic reduction rules. In this setting, it was proved that an evaluation branch which produces the correct result must exist but no effective strategy was presented.

In [15, 6], the lft framework was extended to the extended real line $\mathbb{R}^\infty$ as follows: The leading matrix in a normal product, called a *sign* matrix, has arbitrary integer coefficients. The sign matrix determines an interval of $\mathbb{R}^\infty$ while the subsequent so called *digit* matrices, with all non-negative or all non-positive entries successively refine this interval to the real number in question. In the present paper, we first give a semantics to this framework. We present an extension LCRF (Language for Computable Real Functions) of PCF with a real type which has deterministic reduction rules giving an eventually breadth-first strategy for lazy evaluation of exact real numbers. In the operational semantics, the lazy deterministic strategy allows us to conveniently define $\mathsf{Eval}(P)(n)$ as the nth interval in the output of the real program P.

LCRF is equipped with the constant *redundant if*, rif, which generalizes the quasi-relational comparison operator $<_\epsilon$ with tolerance ϵ in [3, page 47] and which allows us to define a function over the whole of the extended reals by dividing its domain to, say, two overlapping intervals that cover $\mathbb{R}^\infty$. It is decidable in finite time that a real number lies in one of these intervals. If the function is single-valued then its value on the overlap coincides in the two branches of rif and we are able to compute the function thanks to the

*This work has been supported by EPSRC.

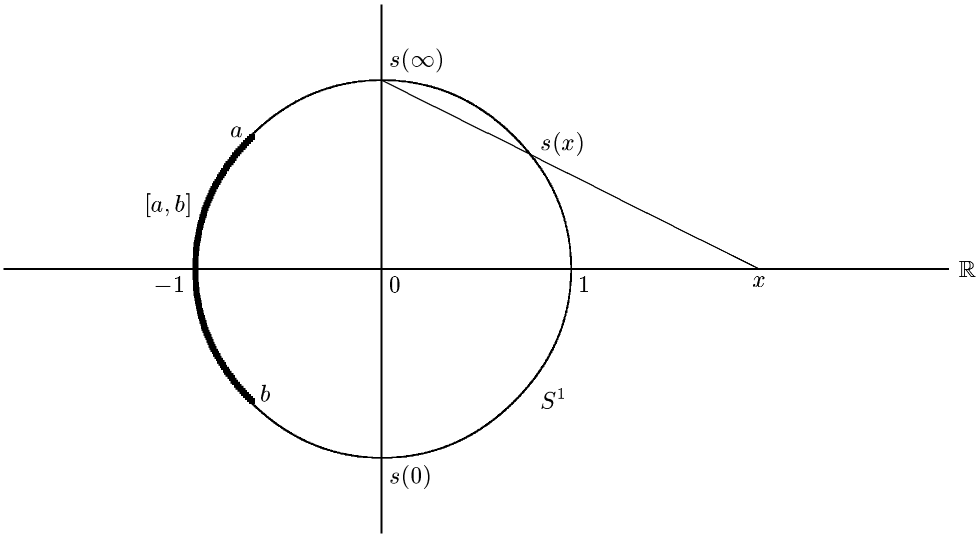


Figure 1: Stereographic projection of the extended real line onto the unit circle and intervals as arcs.

redundancy of our representation. In this way, rif can be regarded as a redundant version of pif used in [4, 7] which, in contrast to pif, overcomes the problem of undecidability of comparison of real numbers in finite time.

We use, for the first time in semantics of programming languages, the upper space $\mathbf{U}\mathbb{R}^\infty$ (the non-empty compact subsets of $\mathbb{R}^\infty$ with reverse inclusion), defined for general topological spaces in [19] and proposed as a computational model for Hausdorff spaces in [5], to develop a denotational semantics for LCRF. In fact, the semantics of rif relies on the existence of the infimum of two approximate reals and this infimum does not exist in the domain $\mathbf{I}\mathbb{R}^\infty$ of intervals of $\mathbb{R}^\infty$. Moreover $\mathbf{U}\mathbb{R}^\infty$ is, in contrast to $\mathbf{I}\mathbb{R}^\infty$, a continuous Scott domain which provides a convenient setting for our semantics.

We finally prove two adequacy results. The first applies to all programs which do not contain rif. The other adequacy result applies to an inductively defined class of so called *basic* real programs which contain rif and include real programs for all single-valued elementary functions.

2 Preliminaries

We denote with $\mathbb{R}^\infty = \mathbb{R} \cup \{\infty\}$ the one point compactification of the real line. Furthermore, we define for $a, b \in \mathbb{R}^\infty$ the *closed interval* from a to b by $[a, b] = \{x \in \mathbb{R}^\infty \mid a \leq x \leq b\}$ for $a \leq b$ and $[a, b] = \{\infty\} \cup \{x \in \mathbb{R} \mid x \leq b\} \cup \{x \in \mathbb{R} \mid a \leq x\}$ for $b < a$. So $[1, -1]$ denotes the complement of the open interval $(-1, 1)$ in $\mathbb{R}^\infty$. If $\mathbb{R}^\infty$ is represented by the unit circle S^1 in the usual way using the stereographic projection, these intervals correspond to arcs going anti-clockwise from a to b, see Figure 1.

The arithmetic operations are extended to $\mathbb{R}^\infty$ in the usual manner, i.e. $x + \infty = \infty + x = \infty - x = x - \infty = \infty$ for $x \neq \infty$ and $x \cdot \infty = \infty \cdot x = \infty$ for $x \neq 0$ as well as $\frac{1}{\infty} = 0$ and $\frac{1}{0} = \infty$. To be able to deal with the remaining cases, we will also consider $\mathbb{R}^\infty_\perp = \mathbb{R}^\infty \cup \{\perp\}$ and set $\infty + \infty = \infty - \infty = 0 \cdot \infty = \frac{0}{0} = \perp$ and $x * \perp = \perp * x = \perp$ for all $x \in \mathbb{R}^\infty_\perp$ and $* \in \{+, -, \cdot, /\}$.

2.1 Real Numbers as Normal Products

We now briefly describe the framework for exact real arithmetic from [15, 6]. A real number $x \in \mathbb{R}^\infty$ is presented as the infinite intersection of a nested sequence of rational intervals, i.e. by a sequence $I_0 \supseteq I_1 \supseteq I_2 \supseteq \cdots$ such that $\bigcap_{n=0}^\infty I_n = \{x\}$. We will consider interval sequences obtained by applying a sequence of compositions of linear fractional transformations with integer coefficients to the base interval $[0, \infty]$ as follows. A (one-dimensional) *linear fractional transformation* (lft) is a function $f \colon \mathbb{R}^\infty_\perp \to \mathbb{R}^\infty_\perp$ where

$$f(x) = \frac{ax + c}{bx + d}$$

for certain parameters $a, b, c, d \in \mathbb{R}$. If $ad - bc \neq 0$ then this function restricts to a homeomorphism of $\mathbb{R}^\infty$ onto itself. Now consider a sequence $f_0, f_1, f_2, \ldots$ of lft's such that the interval sequence defined by

$$\begin{aligned} I_0 &= f_0([0, \infty]) \\ I_1 &= f_0(f_1([0, \infty])) \\ I_2 &= f_0(f_1(f_2([0, \infty]))) \\ &\;\;\vdots \end{aligned} \tag{1}$$

is nested and intersects in a single point. Here, application of an lft to a set is understood pointwise, i.e. we set $f(A) = \{f(x) \mid x \in A\}$ for $A \subseteq \mathbb{R}^\infty$.

We restrict ourselves to lft's with integer coefficients and represent them by matrices. Let us define

$$\mathbb{M} = \left\{ \begin{pmatrix} a & c \\ b & d \end{pmatrix} \;\middle|\; a, b, c, d \in \mathbb{Z} \right\}$$

and for $M \in \mathbb{M}$ let $\Theta(M)$ be the lft associated with M, i.e.

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

Note that $\Theta(MN) = \Theta(M) \circ \Theta(N)$, where MN denotes the usual matrix product. For convenience, we will frequently drop Θ.

The *information* of a matrix M is defined by

$$\mathsf{Info}(M) = M([0,\infty])$$

if the map $\Theta(M)$ does not take the value $\perp$ in $[0,\infty]$. Otherwise, we let $\mathsf{Info}(M) = \mathbb{R}^\infty$. By monotonicity of lft's, it follows that

$$\mathsf{Info}\begin{pmatrix} a & c \\ b & d \end{pmatrix} = \begin{cases} \left[\frac{a}{b}, \frac{c}{d}\right] & \text{if } ad - bc < 0 \\ \left[\frac{c}{d}, \frac{a}{b}\right] & \text{if } ad - bc > 0 \end{cases}$$

and

$$\mathsf{Info}\begin{pmatrix} rp & sp \\ rq & sq \end{pmatrix} = \begin{cases} \left\{\frac{p}{q}\right\} & \text{if } rs > 0 \\ \mathbb{R}^\infty & \text{if } rs < 0. \end{cases}$$

Let $\mathbb{M}^+ = \{\left(\begin{smallmatrix} a & c \\ b & d \end{smallmatrix}\right) \in \mathbb{M} \mid a,b,c,d \geq 0 \text{ or } a,b,c,d \leq 0\}$ denote the set of matrices with same sign integer coefficients. Note that for matrices M with non-zero columns we have $\mathsf{Info}(M) \subseteq [0,\infty]$ iff $M \in \mathbb{M}^+$. In particular, this implies that $\mathsf{Info}(NM) \subseteq \mathsf{Info}(N)$ for such M and all $N \in \mathbb{M}$. Thus picking $f_0 = \Theta(S)$ and $f_n = \Theta(M_n)$ for $n \geq 1$ with matrices $S \in \mathbb{M}$ and $M_n \in \mathbb{M}^+$ results in a shrinking interval sequence as defined in (1). This is the basic idea of how to represent elements of $\mathbb{R}^\infty$ by sequences of integer matrices.

This idea is refined by introducing *vectors* which enable us to represent rational numbers exactly. Let

$$\mathbb{V} = \left\{ \begin{pmatrix} a \\ b \end{pmatrix} \;\middle|\; a, b \in \mathbb{Z} \right\}$$

the set of vectors with integer coefficients. The zero-dimensional lft or the constant associated with a vector is defined by $\Theta\left(\begin{smallmatrix} a \\ b \end{smallmatrix}\right) = \frac{a}{b} \in \mathbb{R}^\infty_\perp$. We will also consider the set $\mathbb{V}^+ = \{\left(\begin{smallmatrix} a \\ b \end{smallmatrix}\right) \in \mathbb{V} \mid ab \geq 0\}$ of vectors with same sign integer coefficients which correspond to rational numbers in $[0,\infty]$ or to $\perp$.

Now we are able to define our representation of real numbers. First, the special case of elements of $[0,\infty]$ is treated separately.

Definition 1 *An* unsigned normal product *is either an infinite sequence of matrices of* $\mathbb{M}^+$*, called digit matrices, or a finite list of such matrices terminated by a vector from* $\mathbb{V}^+$*. The unsigned normal product* $\alpha = M_0 M_1 M_2 M_3 \cdots$ *represents the interval sequence* $\mathsf{seq}(\alpha)$ *with*

$$\mathsf{seq}(\alpha)(n) = [0,\infty] \cap \bigcap_{i=0}^{n} \mathsf{Info}(M_0 M_1 M_2 \cdots M_i).$$

If $\alpha = M_0 M_1 \cdots M_n V$ *then* $\mathsf{seq}(\alpha) = \mathsf{seq}(\alpha')$ *with* $\alpha' = M_0 M_1 M_2 \cdots M_n M'_{n+1} M'_{n+2} \cdots$ *where* $M'_{n+1} = (V, V)$ *and* $M'_k = \left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right)$ *for* $k \geq n+2$.

The most common case of an unsigned normal product is that of an infinite sequence $\alpha = M_0 M_1 \cdots$ of matrices which are all non-singular. In this case the associated interval sequence is simply given by

$$\mathsf{seq}(\alpha)(n) = \mathsf{Info}(M_0 M_1 M_2 \cdots M_n).$$

Note that without any further assumptions this interval sequence might converge to an interval with non-empty interior rather than to a single point.

One way to derive unsigned normal products is via continued fractions [21]. For example, take the expansion

$$\sqrt{2} = 1 + \cfrac{1}{2 + \cfrac{1}{2 + \cfrac{1}{2 + \cdots}}}$$

This yields $\sqrt{2} = \frac{x+1}{x}$ with x satisfying

$$x = 2 + \frac{1}{x} = \begin{pmatrix} 2 & 1 \\ 1 & 0 \end{pmatrix} x.$$

Thus we get the unsigned normal product

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} 2 & 1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} 2 & 1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} 2 & 1 \\ 1 & 0 \end{pmatrix}\cdots$$

for $\sqrt{2}$. The corresponding interval sequence starts with

$$[1,\infty] \supseteq [1, 1.5] \supseteq [1.4, 1.5] \supseteq [1.4, \tfrac{17}{12}] \supseteq \cdots.$$

Allowing the first matrix in a normal product to be arbitrary, we get a representation of all of $\mathbb{R}^\infty$.

Definition 2 *A* signed normal product *consists either of a* sign matrix *from* $\mathbb{M}$ *followed by an unsigned normal product or of a single vector from* $\mathbb{V}$*. The signed normal product* $\alpha = M_0 M_1 M_2 M_3 \cdots$ *represents the interval sequence* $\mathsf{seq}(\alpha)$ *with*

$$\mathsf{seq}(\alpha)(n) = \bigcap_{i=0}^{n} \mathsf{Info}(M_0 M_1 M_2 \cdots M_i).$$

If $\alpha = M_0 M_1 \cdots M_n V$ *then* $\mathsf{seq}(\alpha) = \mathsf{seq}(\alpha')$ *with* $\alpha' = M_0 M_1 M_2 \cdots M_n M'_{n+1} M'_{n+2} \cdots$ *where* $M'_{n+1} = (V, V)$ *and* $M'_k = \left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right)$ *for* $k \geq n+2$.

Note that if $\beta = M_0 M_1 M_2 M_3 \cdots = M_0 \alpha$ is a signed normal product then α is an unsigned normal product and $\mathsf{seq}(\beta)(n) = M_0(\mathsf{seq}(\alpha)(n))$ for $n \geq 1$. As in the case of unsigned normal products, the most common case of a signed normal product is that of an infinite sequence $\alpha = M_0 M_1 \cdots$ of matrices which are all non-singular. In this case the associated interval sequence is simply given by

$$\mathsf{seq}(\alpha)(n) = \mathsf{Info}(M_0 M_1 M_2 \cdots M_n).$$

Extending the above example, an unsigned normal product for $\frac{1-\sqrt{2}}{1+\sqrt{2}}$ is given by

$$\begin{pmatrix} -1 & 1 \\ 1 & 1 \end{pmatrix}\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} 2 & 1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} 2 & 1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} 2 & 1 \\ 1 & 0 \end{pmatrix}\cdots.$$

Finally, we need one more definition: The *pseudo-inverse* of a non-singular matrix $M = \left(\begin{smallmatrix} a & c \\ b & d \end{smallmatrix}\right)$ is given by $\widetilde{M} = \left(\begin{smallmatrix} d & -c \\ -b & a \end{smallmatrix}\right)$. Then $\Theta(M\widetilde{M}) = \Theta(\widetilde{M}M)$ is the identity. Note that for non-singular M, N we have $\mathsf{Info}(N) \subseteq \mathsf{Info}(M)$ iff $\widetilde{M}N \in \mathbb{M}^+$. In this case, we want the entries of this matrix to be non-negative and hence define

$$M^*N = \begin{cases} -\widetilde{M}N & \text{if all entries of } \widetilde{M}N \text{ are non-positive} \\ \widetilde{M}N & \text{otherwise} \end{cases}$$

for all matrices $M, N \in \mathbb{M}$. Then $M^*N \in \mathbb{M}^+$ if and only if $\mathsf{Info}(N) \subseteq \mathsf{Info}(M)$ and the entries of M^*N are non-negative exactly in this case.

2.2 Tensors and Expression Trees

The framework of [15, 6] employs the set

$$\mathbb{T} = \left\{ \begin{pmatrix} a & c & e & g \\ b & d & f & h \end{pmatrix} \;\middle|\; a, b, c, d, e, f, g \in \mathbb{Z} \right\}$$

of *tensors* to capture functions. The two-dimensional lft assigned to a tensor is defined by

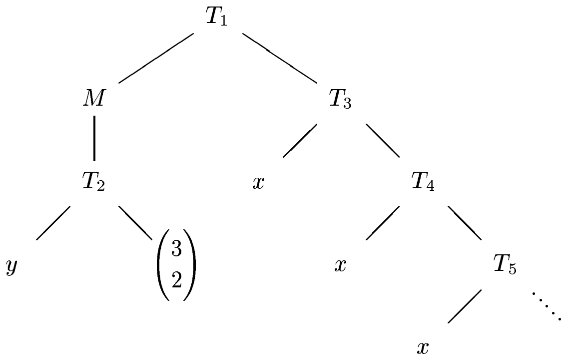
$$\Theta \begin{pmatrix} a & c & e & g \\ b & d & f & h \end{pmatrix} : (x, y) \mapsto \frac{axy + cx + ey + g}{bxy + dx + fy + h} : (\mathbb{R}_\perp^\infty)^2 \to \mathbb{R}_\perp^\infty.$$

Simple examples are tensors for the basic arithmetic operations, e.g.

$$T_+ = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad \text{and} \qquad T_\times = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

for addition and multiplication, respectively.

Compound computations and the elementary functions are computed using possibly infinite binary trees whose nodes are either tensors with two children, matrices with one child, or vectors with no children. Normal products are the simplest possible such trees. Another example is the tree[1]

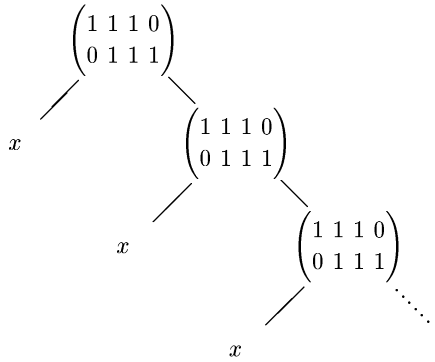


which denotes the infinite expression

$$T_1\Big(MT_2(y, 1.5), T_3(x, T_4(x, T_5(x, \ldots)))\Big),$$

where x and y are real numbers given as normal products. It is shown in [13] that for every elementary function there exists such a tree which computes the function in the lft framework. We give in Section 3.3 below an example for the tangent function.

We derive an interval sequence from such a tree in the following manner: For the nth interval, cut the tree off at depth n and plug in the interval $[0, \infty]$ for all exposed arguments. The resulting intervals will be nested, if all lft's in the tree have non-negative entries except possibly for the root node. We call such a tree a *signed expression tree.* For the special case that the entries of all tensors, matrices and vectors in the tree are non-negative, we speak of an *unsigned expression tree.* If there are just finitely many nodes with negative entries, we are dealing with *expression trees.* Before we elaborate how to assign interval sequences to expression trees, we give one more example from [13]. The tree



can be shown to evaluate $\sqrt{x}$ for x given as an unsigned normal product.

Evaluating an expression in this framework amounts to transforming the expression tree into an equivalent tree which is a normal product. We need some further definitions in order to explain the possible transformations. The *transpose* of a tensor is defined by $\left(\begin{smallmatrix} a & c & e & g \\ b & d & f & h \end{smallmatrix}\right)^{\mathsf{t}} = \left(\begin{smallmatrix} a & e & c & g \\ b & f & d & h \end{smallmatrix}\right)$. Furthermore, $(T_1, T_2) \circledR M = (T_1 \cdot M, T_2 \cdot M)$ is the *right product* of a tensor with a matrix; the left product is defined by $T \,Ⓛ\, M = (T^{\mathsf{t}} \circledR M)^{\mathsf{t}}$. Here we write T_1, respectively T_2, for the matrix consisting of the first, respectively last, two columns of T and, as above, denote the usual matrix product by $\cdot$ or simply juxtaposition. In particular, $M \cdot (T_1, T_2) = (MT_1, MT_2)$. Observe that

$$\begin{aligned} T(M(x), y) &= (T \,Ⓛ\, M)(x, y) && (2) \\ T(x, M(y)) &= (T \circledR M)(x, y) && (3) \\ (M \cdot T)(x, y) &= M(T(x, y)) && (4) \end{aligned}$$

for $T \in \mathbb{T}$ and $M \in \mathbb{M}$ and all $x, y \in \mathbb{R}_\perp^\infty$. These identities are read from left to right as rewrite rules. Here a step of the form $T(M(x), y) \rightsquigarrow (T \,Ⓛ\, M)(x, y)$ or $T(x, M(y)) \rightsquigarrow (T \circledR M)(x, y)$ using (2) or (3) is called *left-absorption* or *right-absorption*, respectively, while $T(x, y) \rightsquigarrow M(T'(x, y))$ derived from (4) is referred to as *emission.* The rules are repeatedly applied to an expression tree such that the root emits a sequence of matrices—the normal product which represents the result of the computation.

The key steps are the emission steps. In order to apply (4), a tensor T has to be split into a product MT'. This has to be done in such a way that the remaining tensor T' has positive entries in order to ensure that the resulting sequence of matrices is a normal product, i.e. that the associated intervals form a nested sequence. This is achieved by considering the information of a tensor and emitting a matrix with exactly the same information contents as follows.

The *information* of a tensor T lft is defined by $\mathsf{Info}(T) = T([0, \infty], [0, \infty])$ if $\Theta(T)$ is nonsingular in $[0, \infty]^2$. Otherwise, we let $\mathsf{Info}(T) = \mathbb{R}^\infty$. By monotonicity of lft's, it follows that $\mathsf{Info}(T) = \mathsf{Info}(T_1) \cup \mathsf{Info}(T_2) \cup \mathsf{Info}((T^{\mathsf{t}})_1) \cup \mathsf{Info}((T^{\mathsf{t}})_2)$.

For the following procedure to calculate the information of a tensor, we define the relations $\preceq$ and $\asymp$ on $\mathbb{V}$ by $\binom{a}{b} \preceq \binom{c}{d}$ iff $\binom{a}{b} = \binom{c}{d}$ or $ad - bc < 0$ and $V \asymp W$ iff there is $r < 0$ with $V = rW$.

Definition 3 *Suppose $T = (V_1 V_2 V_3 V_4) \in \mathbb{T}$, where the V_i denote the columns of T. The* head T^{hd} *of T is the matrix*

[1] The trees were typeset using Paul Taylor's diagram macros.

defined as follows: If there are $i, j \in \{1,2,3,4\}$ *with* $V_i \asymp V_j$ *then we set* $T^{\mathsf{hd}} = \left(\begin{smallmatrix}1 & 0\\ 0 & 1\end{smallmatrix}\right)$. *Otherwise we distinguish two cases: (1) if* $\preceq$ *is not a partial order on* $\{V_1, V_2, V_3, V_4\}$, *i.e. if there are pairwise disjoint* i, j, k *with* $V_i \preceq V_j \preceq V_k$, *then we also set* $T^{\mathsf{hd}} = \left(\begin{smallmatrix}1 & 0\\ 0 & 1\end{smallmatrix}\right)$ *and (2) if* $\preceq$ *is a partial order on* $\{V_1, V_2, V_3, V_4\}$ *we define* $T^{\mathsf{hd}} = (\max(V_i), \min(V_i)) \in \mathbb{M}$ *(the* head *of* T*), where we pick the unique maximal/minimal element* $\left(\begin{smallmatrix}a\\ b\end{smallmatrix}\right)$ *with minimal sum* $a+b$.

The tail T^{tl} *of* T *is the tensor defined by* $T^{\mathsf{tl}} = \widetilde{T^{\mathsf{hd}}}T \in \mathbb{T}$.

Lemma 4 *For any* $T \in \mathbb{T}$, *head and tail as given above are well-defined. If* $\mathsf{Info}(T) \neq \mathbb{R}^\infty$ *then* $\mathsf{Info}(T) = \mathsf{Info}(T^{\mathsf{hd}})$ *and* $T^{\mathsf{tl}} \in \mathbb{T}^+$. *If* $\mathsf{Info}(T) = \mathbb{R}^\infty$ *then* $T^{\mathsf{hd}} = \left(\begin{smallmatrix}1 & 0\\ 0 & 1\end{smallmatrix}\right)$.

With this tool at hand we are able to transform nondegenerate expression trees into signed expression trees in a finite number of steps.

Theorem 5 *Every expression tree with the property that all its unsigned subtrees denote single points can be transformed into a signed expression tree with a finite number of emissions and absorptions.*

Proof. Starting with the lowermost signed tensor T, we absorb from both sides until $\mathsf{Info}(T) \neq \mathbb{R}^\infty$. Then we are able to emit a signed matrix; the remaining tensor is unsigned. Signed matrices and vectors are absorbed. As there is just a finite number of signed nodes, after a finite number of step the root is the only node left which might contain negative entries. The new tree is a signed expression tree. ■

3 LCRF

The Language for Computable Real Functions (LCRF) includes the syntax and conventions of the Programming Language for Computable Functions (PCF) with products described by Gunter [8]. This in turn includes the terms of the simply-typed λ-calculus.

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

$$
\begin{array}{rcl}
x & \in & \mathsf{variable} \\
t & ::= & \mathsf{num} \mid \mathsf{bool} \mid \mathsf{real} \mid \mathsf{real}^+ \mid t \to t \mid t \times t \\
P & ::= & x \mid \mathbf{0} \mid \mathsf{true} \mid \mathsf{false} \mid \\
 & & \mathsf{succ}\,(P) \mid \mathsf{pred}\,(P) \mid \mathsf{zero}\,(P) \mid \\
 & & \mathsf{if}\,P\,\mathsf{then}\,P\,\mathsf{else}\,P \mid \\
 & & \lambda x : t.P \mid PP \mid \mu x : t.P \mid \\
 & & (P, P) \mid \mathsf{fst}\,(P) \mid \mathsf{snd}\,(P) \mid \\
 & & \mathsf{rif} \mid [P] \mid [P\rangle \mid \langle P\rangle \mid \rangle P\rangle
\end{array}
$$

where $\mathsf{variable}$ 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, real and real^+ are called the *ground* types. Due to the special role of the interval $[0, \infty]$ in the theory of lft's, we have two types corresponding to the reals, real for $\mathbb{R}^\infty$ and real^+ for $[0, \infty]$.

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 – $\mathsf{succ}(P)$, predecessor – $\mathsf{pred}(P)$, test for zero – $\mathsf{zero}(P)$, conditional – $\mathsf{if}\,P\,\mathsf{then}\,P\,\mathsf{else}\,P$, pairing – (P, Q), first projection – $\mathsf{fst}(P)$, second projection – $\mathsf{snd}(P)$ and recursion – $\mu x : t.P$. The new constructs for LCRF are *redundant if* – rif, used for definition of real functions by cases, and *transformations* — $[P]$, $[P\rangle$, $\langle P\rangle$ and $\rangle P\rangle$, corresponding to Θ. As we will see later, the different types of brackets correspond to different types of input and output.

The equivalence class of term trees modulo renaming of bound variables are called just *terms* and we refer to a closed term of a type which is a product of ground types as a *program.* 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, int denotes an integer constructed in the usual way using a pair of natural numbers num, $\mathsf{vector}(t)$ denotes $t \times t$, $\mathsf{matrix}(t)$ denotes $\mathsf{vector}(t) \times \mathsf{vector}(t)$ and $\mathsf{tensor}(t)$ denotes $\mathsf{matrix}(t) \times \mathsf{matrix}(t)$.

There are two systems of rules describing LCRF. 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.

3.1 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 [8], and

$$
\text{[Vec1]}\ \frac{H \vdash V : \mathsf{vector(int)}}{H \vdash [V\rangle : \mathsf{real}}
\qquad
\text{[Vec2]}\ \frac{H \vdash V : \mathsf{vector\,(num)}}{H \vdash \langle V\rangle : \mathsf{real}^+}
$$

$$
\text{[Mat1]}\ \frac{H \vdash M : \mathsf{matrix\,(int)}}{H \vdash [M] : \mathsf{real} \to \mathsf{real}}
\qquad
\text{[Mat2]}\ \frac{H \vdash M : \mathsf{matrix\,(int)}}{H \vdash [M\rangle : \mathsf{real}^+ \to \mathsf{real}}
\qquad
\text{[Mat3]}\ \frac{H \vdash M : \mathsf{matrix\,(num)}}{H \vdash \langle M\rangle : \mathsf{real}^+ \to \mathsf{real}^+}
$$

$$
\text{[Ten1]}\ \frac{H \vdash T : \mathsf{tensor\,(int)}}{H \vdash [T] : \mathsf{real} \times \mathsf{real} \to \mathsf{real}}
\qquad
\text{[Ten2]}\ \frac{H \vdash T : \mathsf{tensor\,(int)}}{H \vdash [T\rangle : \mathsf{real}^+ \times \mathsf{real}^+ \to \mathsf{real}}
$$

$$
\text{[Ten3]}\ \frac{H \vdash T : \mathsf{tensor\,(num)}}{H \vdash \langle T\rangle : \mathsf{real}^+ \times \mathsf{real}^+ \to \mathsf{real}^+}
\qquad
\text{[Ten4]}\ \frac{H \vdash T : \mathsf{tensor(num)}}{H \vdash \rangle T\rangle : \mathsf{real}^+ \times \mathsf{real}^+ \to \mathsf{real}^+}
$$

$$
\text{[Rif]}\ \frac{\begin{array}{c} H \vdash F : \mathsf{real}^+ \to \mathsf{real} \\ H \vdash G : \mathsf{real}^+ \to \mathsf{real} \\ M \in \mathbb{M} \quad N \in \mathbb{M} \\ \mathsf{Info}(M)^\circ \cup \mathsf{Info}(N)^\circ = \mathbb{R}^\infty \\ \mathsf{Info}(M) \neq \mathbb{R}^\infty \neq \mathsf{Info}(N) \end{array}}{H \vdash \mathsf{rif}(M, N, F, G) : \mathsf{real} \to \mathsf{real}}
$$

$$\langle\text{Mat1}\rangle\quad \frac{P \to [N\rangle Q}{[M]P \to [MN\rangle Q}$$

$$\langle\text{Mat2}\rangle\quad \frac{P \to [N]Q}{[M]P \to [MN]Q}$$

$$\langle\text{Emit1}\rangle\quad \langle T\rangle(P,Q) \to \langle T^{\mathsf{hd}}\rangle\, \rangle T^{\mathsf{tl}}\rangle(P,Q)$$

$$\langle\text{Emit2}\rangle\quad \frac{\mathsf{info}\,(T) \neq \mathbb{R}^\infty}{[T\rangle(P,Q) \to [T^{\mathsf{hd}}\rangle\, \rangle T^{\mathsf{tl}}\rangle(P,Q)}$$

$$\langle\text{Abs1}\rangle\quad \frac{P \to \langle M\rangle P' \quad Q \to \langle N\rangle Q'}{\rangle T\rangle(P,Q) \to \langle T \circledL\, M \circledR\, N\rangle(P',Q')}$$

$$\langle\text{Abs2}\rangle\quad \frac{\mathsf{Info}(T) = \mathbb{R}^\infty \quad P \to \langle M\rangle P' \quad Q \to \langle N\rangle Q'}{[T\rangle(P,Q) \to [T \circledL\, M \circledR\, N\rangle(P',Q')}$$

$$\langle\text{Abs3}\rangle\quad \frac{P \to [M\rangle P' \quad Q \to [N\rangle Q'}{[T](P,Q) \to [T \circledL\, M \circledR\, N\rangle(P',Q')}$$

$$\langle\text{Rif1}\rangle\quad \frac{M^*O \in \mathbb{M}^+}{\mathsf{rif}(M,N,F,G)[O\rangle P \to F(\langle M^*O\rangle P)}$$

$$\langle\text{Rif2}\rangle\quad \frac{M^*O \notin \mathbb{M}^+ \quad N^*O \in \mathbb{M}^+}{\mathsf{rif}(M,N,F,G)[O\rangle P \to G(\langle N^*O\rangle P)}$$

$$\langle\text{Rif3}\rangle\quad \frac{M^*O \notin \mathbb{M}^+ \quad N^*O \notin \mathbb{M}^+ \quad P \to \langle O'\rangle Q}{\mathsf{rif}(M,N,F,G)([O\rangle P) \to \mathsf{rif}(M,N,F,G)([OO'\rangle Q)}$$

Figure 2: Reduction rules for LCRF.

where A° denotes the interior of $A \subseteq \mathbb{R}^\infty$.

As can be seen from these rules, we use the different types of brackets to denote different types of input and output to a function. So the right square bracket ']' denotes input type real and the left square bracket '[' output type real while right angle bracket '⟩' stands for input real$^+$ and left angle bracket '⟨' for output real$^+$. For simplicity, we do not consider functions with input real and output real$^+$, hence there is no transformation $\langle P]$. The special construct $\rangle T\rangle$ is used to indicate that a tensor is in a state where only absorption is allowed.

The restrictions in the rule [Rif] ensure that the two cases cover all input values and overlap. Intuitively, the term $\mathsf{rif}(M,N,f,g)$ should be thought of as the function $x \mapsto f(M^{-1}x)$ for $x \in \mathsf{Info}(M)$ and $x \mapsto g(N^{-1}x)$ for $x \in \mathsf{Info}(N)$.

3.2 Reduction Rules

A real program can be considered as an expression tree as described in Section 2.2. The reduction rules will include the rules of PCF and allow expression trees to be transformed into normal products by absorption and emission rules. Formally, the reduction rules of LCRF are defined in two steps. First we extend the reduction relation $\to$ of PCF to our language. This relation allows us to emit a leading matrix from any expression tree, i.e. the sign matrix for type real and the first digit matrix for type real$^+$. Then we define the relation $\Longrightarrow$ which enables us to emit sequences of matrices.

We define $\to$ to be the least reflexive transitive relation satisfying the reduction rules for *call-by-name evaluation* of PCF with products, which is well known [8], and the rules given in Figure 2, where, for simplicity, we have omitted a corresponding set of rules for vectors in place of matrices.

The rules ⟨Mat1/2⟩ ensure that only the leading matrix of a normal product may have negative entries. For the same reason, a tensor $[T]$ may only absorb information ⟨Abs3⟩ and a tensor $[T\rangle$ may only emit if it has non-trivial information ⟨Emit2⟩ and has to absorb otherwise ⟨Abs2⟩. Whenever a tensor emits information ⟨Emit1/2⟩, its state is changed to $\rangle T\rangle$, so that absorption is forced next ⟨Abs1⟩. This ensures that in an infinite tree $\langle T_1\rangle(x, \langle T_2\rangle(x, \langle T_3\rangle(x, \ldots)))$ which can be constructed using μ (see the example in 3.3 below), information is processed in a breadth-first manner to avoid possible unnecessary deadlock.

Note that the rules for rif are deterministic. Priority is given to the left branch ⟨Rif1⟩, the right branch is chosen only if a left-choice is not possible ⟨Rif2⟩. Only if neither branch can be chosen yet, the argument is further reduced ⟨Rif3⟩.

The *reduction relation* $\Longrightarrow$ is now defined as the least relation containing $\to$ and satisfying

$$\text{(i)}\quad \frac{P \Longrightarrow Q}{[M\rangle P \Longrightarrow [M\rangle Q}$$

$$\text{(ii)}\quad \frac{P \Longrightarrow Q}{\langle M\rangle P \Longrightarrow \langle M\rangle Q}.$$

Hence $\Longrightarrow$ allows us to emit sequences of matrices. The distinction between $\to$ and $\Longrightarrow$ is necessary to ensure that the rules are deterministic: Suppose $P \to \langle M\rangle P'$ and $P' \to \langle M'\rangle P''$ as well as $Q \to \langle N\rangle Q'$. Then there is only one way of reducing $\rangle T\rangle(P,Q)$, namely to $\langle T \circledL\, M \circledR\, N\rangle(P',Q')$ by ⟨Abs1⟩. If we did not distinguish between $\to$ and $\Longrightarrow$, then (ii) would yield $P \to \langle M\rangle\langle M'\rangle P''$ and the reduction $\rangle T\rangle(P,Q) \to \langle T \circledL\, M \circledR\, N\rangle(\langle M'\rangle P'', Q)$ would also be possible.

We say that a program P evaluates to a value Z if $P \Longrightarrow Z$ where a value of type bool, num, or int is, as usual, a constant or of the form $\mathsf{succ}^n(0)$ for some $n \in \mathbb{N}$, respectively, so that no further reduction is possible. As usual, values of product type are tuples with values as components. For type real, values are programs of the form $[V\rangle$ or

$$[M_0\rangle\langle M_1\rangle \cdots \langle M_n\rangle P$$

for some program P and matrices with values as entries. Similarly for type real$^+$ with $\langle M_0\rangle$ as leading term. Note that, unlike the case of the other types, a value of type real or I^+ may be further reduced unless it is a vector.

Even though reduction for the new types need not terminate, successive reduction yields a sequence of matrices as the rules are deterministic. Thus we are able to extend the evaluation function Eval from programs to values to the

$$
\begin{aligned}
\mathsf{tan}\left(\frac{x-1}{x+1}\right) &= [J\rangle(x,\Phi(0)) \\
&= \left[\begin{matrix}1&1&-1&-1\\2&0&0&2\end{matrix}\right\rangle (x,\Phi\,(0)) \\
&\Longrightarrow \left[\begin{matrix}1&1&-1&-1\\2&0&0&2\end{matrix}\right\rangle (x,\lambda n.\,\langle I\,(n)\rangle\,(x,\Phi\,(n+1))\,0) \\
&\Longrightarrow \left[\begin{matrix}1&1&-1&-1\\2&0&0&2\end{matrix}\right\rangle (x,\langle I\,(0)\rangle\,(x,\Phi\,(1))) \\
&\Longrightarrow \left[\begin{matrix}1&1&-1&-1\\2&0&0&2\end{matrix}\right\rangle \left(x,\left\langle\begin{matrix}3&1&3&5\\5&3&1&3\end{matrix}\right\rangle(x,\Phi\,(1))\right) \\
&\Longrightarrow \left[\begin{matrix}1&1&-1&-1\\2&0&0&2\end{matrix}\right\rangle \left(x,\left\langle\begin{matrix}3&1&3&5\\5&3&1&3\end{matrix}\right\rangle(x,\lambda n.\,\langle I\,(n)\rangle\,(x,\Phi\,(n+1))\,1)\right) \\
&\Longrightarrow \left[\begin{matrix}1&1&-1&-1\\2&0&0&2\end{matrix}\right\rangle \left(x,\left\langle\begin{matrix}3&1&3&5\\5&3&1&3\end{matrix}\right\rangle(x,\langle I\,(1)\rangle\,(x,\Phi\,(2)))\right) \\
&\Longrightarrow \left[\begin{matrix}1&1&-1&-1\\2&0&0&2\end{matrix}\right\rangle \left(x,\left\langle\begin{matrix}3&1&3&5\\5&3&1&3\end{matrix}\right\rangle\left(x,\left\langle\begin{matrix}5&3&5&7\\7&5&3&5\end{matrix}\right\rangle(x,\Phi\,(2))\right)\right)
\end{aligned}
$$

Figure 3: Unfolding the expression tree for $\mathsf{tan}(\frac{x-1}{x+1})$ where $x \in [0,\infty]$ does not reduce.

new types as follows. If the program P reduces to values starting with arbitrarily many matrices, it determines a normal product α, signed or unsigned depending on the type. We let $\mathsf{Eval}(P) = \mathsf{seq}(\alpha)$ be the sequence of information thus calculated. Thus, e.g., if P is of type real and $P \Longrightarrow [M_0\rangle\langle M_1\rangle\cdots\langle M_n\rangle Q$ and all these matrices are non-singular then $\mathsf{Eval}(P)(n) = \mathsf{Info}(\prod_{i=0}^{n} M_i)$.

3.3 Example

A continued fraction for tangent is (Lambert [9])

$$\mathsf{tan}(x) = \cfrac{1}{\frac{1}{x} + \cfrac{1}{-\frac{3}{x} + \cfrac{1}{\frac{5}{x} + \cfrac{1}{-\frac{7}{x} + \cdots}}}}$$

This can be transformed [13] to

$$\mathsf{tan}(\tfrac{x-1}{x+1}) = \begin{pmatrix}x-1 & x-1\\ 2x & 2\end{pmatrix}\prod_{n=0}^{\infty}\begin{pmatrix}(2n+3)(x+1) & (2n+1)x+(2n+5)\\ (2n+5)x+(2n+1) & (2n+3)(x+1)\end{pmatrix}$$

or put another way

$$\mathsf{tan}(\left(\begin{smallmatrix}1&-1\\1&1\end{smallmatrix}\right)x) = \left(\begin{smallmatrix}1&1&-1&-1\\2&0&0&2\end{smallmatrix}\right)(x,\left(\begin{smallmatrix}3&1&3&5\\5&3&1&3\end{smallmatrix}\right)(x,\left(\begin{smallmatrix}5&3&5&7\\7&5&3&5\end{smallmatrix}\right)(x,\cdots)))$$

for $x \in [0,\infty]$.

This infinite tree can be generated using the μ construct. If we set

$$I = \lambda n.\begin{pmatrix}2n+3 & 2n+1 & 2n+3 & 2n+5\\ 2n+5 & 2n+3 & 2n+1 & 2n+3\end{pmatrix}$$

and

$$\Phi = \mu f.\lambda n.\,\langle I\,(n)\rangle\,(x, f\,(n+1))$$

then

$$\Phi \to \lambda n.\langle I(n)\rangle(x,\Phi(n+1)),$$

so

$$\Phi(0) \to \langle I(0)\rangle(x,\Phi(1)) \to \langle I(0)\rangle(x,\langle I(1)\rangle(x,\Phi(2))) \to \cdots$$

provided x does not reduce. Thus, a program $h\colon \mathsf{real}^+ \to \mathsf{real}$ in LCRF with $h(x) = \mathsf{tan}(\left(\begin{smallmatrix}1&-1\\1&1\end{smallmatrix}\right)x)$ for $x \in [0,\infty]$ is

$$h(x) = [J\rangle(x,\Phi(0))$$

where $J = \left(\begin{smallmatrix}1&1&-1&-1\\2&0&0&2\end{smallmatrix}\right)$. The first few unfoldings of h are displayed in Figure 3.

This enables us to calculate $\mathsf{tan}(x)$ for all $x \in [-1,1]$. For $x \notin [-1,1]$, we must repeatedly apply the trigonometric identity

$$\mathsf{tan}\,(x) = \frac{2\,\mathsf{tan}\left(\frac{x}{2}\right)}{1-\mathsf{tan}\left(\frac{x}{2}\right)^2} = [T]\left(\mathsf{tan}\left(\tfrac{x}{2}\right), \mathsf{tan}\left(\tfrac{x}{2}\right)\right)$$

where $T = \left(\begin{smallmatrix}0&1&1&0\\-1&0&0&1\end{smallmatrix}\right)$ in order to shift the argument of tan into $[-1,1]$. A recursive application of the rif-statement can be used to accomplish this provided we use a suitable pair of overlapping intervals. For instance, consider the two matrices $M = \left(\begin{smallmatrix}1&-1\\1&1\end{smallmatrix}\right)$ and $N = \left(\begin{smallmatrix}1&1\\-2&2\end{smallmatrix}\right)$ with $\mathsf{Info}(M) = [-1,1]$ and $\mathsf{Info}(N) = [\frac{1}{2},-\frac{1}{2}]$. We get

$$\mathsf{tan} = \mu f.\mathsf{rif}(M,N,h,\lambda x.[T](f([N'\rangle x), f([N'\rangle x)))$$

where $N' = \left(\begin{smallmatrix}1&1\\-4&4\end{smallmatrix}\right)$.

So, if the term tan is applied to a real number, $[L\rangle\, x$ say, then if $\mathsf{Info}(L) \subseteq \mathsf{Info}(M) = [-1,1]$ the result is

$$
\begin{aligned}
h\,(\langle M^* L\rangle\, x) &= \mathsf{tan}\left(\left(\begin{smallmatrix}1&-1\\1&1\end{smallmatrix}\right)M^* L\,(x)\right) \\
&= \mathsf{tan}\,(L\,(x))
\end{aligned}
$$

else if $\mathsf{Info}(L) \subseteq \mathsf{Info}(N) = [\frac{1}{2},-\frac{1}{2}]$ the result is

$$
\begin{aligned}
&[T]\left(\mathsf{tan}\left([N'\rangle\,\langle N^* L\rangle\, x\right), \mathsf{tan}\left([N'\rangle\,(\langle N^* L\rangle\, x)\right)\right) \\
&= \begin{bmatrix}0&1&1&0\\-1&0&0&1\end{bmatrix}\left(\mathsf{tan}\left([N' N^* L\rangle\, x\right), \mathsf{tan}\left([N' N^* L\rangle\, x\right)\right)
\end{aligned}
$$

$$= \begin{bmatrix} 0 & 1 & 1 & 0 \\ -1 & 0 & 0 & 1 \end{bmatrix} \left(\tan\left([\left(\begin{smallmatrix}4&0\\0&8\end{smallmatrix}\right)L\rangle\, x\right), \tan\left([\left(\begin{smallmatrix}4&0\\0&8\end{smallmatrix}\right)L\rangle\, x\right)\right)$$

$$= \frac{2\tan\left(\frac{L(x)}{2}\right)}{1-\tan\left(\frac{L(x)}{2}\right)^2},$$

otherwise $[L\rangle\, x$ must be evaluated further.

4 Denotational Semantics

We refer the reader to [1] for domain theory. Let $\mathbb{N}_\perp = \mathbb{N}\cup\{\perp\}$ and $\mathbb{B}_\perp = \{\mathsf{true}, \mathsf{false}, \perp\}$ denote the flat domains of natural numbers and truth values, respectively. For a compact Hausdorff space X, let $\mathbf{U}X$ be its upper power space, i.e. the collection of all nonempty closed subsets of X, ordered by reversed set inclusion. The space X is embedded into $\mathbf{U}X$ as the set of maximal elements via $x \mapsto \{x\}$. The way-below relation $\ll$ on $\mathbf{U}X$ is given by $K \ll L$ iff there is an open set $O \subseteq X$ such that $K \supseteq O \supseteq L$. If $f\colon X \to Y$ is continuous, its canonical extension $\mathbf{U}f\colon \mathbf{U}X \to \mathbf{U}Y$ is given by $\mathbf{U}f(A) = \{f(x) \mid x \in A\}$. We write $\mathbf{U}f$ simply as f. We will use these domains, which are all continuous Scott-domains, i.e. are bounded complete, to extend the *standard fixed-point model* relative to call-by-name of PCF with products to LCRF. The interpretation $[\![t]\!]$ of a type t is the domain defined inductively by

$$\begin{array}{rcl@{\qquad}rcl} [\![\mathsf{num}]\!] &=& \mathbb{N}_\perp & [\![\mathsf{real}^+]\!] &=& \mathbf{U}[0,\infty] \\ [\![\mathsf{bool}]\!] &=& \mathbb{B}_\perp & [\![s \to t]\!] &=& [[\![s]\!] \to [\![t]\!]] \\ [\![\mathsf{real}]\!] &=& \mathbf{U}\mathbb{R}^\infty & [\![s \times t]\!] &=& [\![s]\!] \times [\![t]\!] \end{array}$$

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, in particular, we get for transformations that

$$\begin{array}{rcl} [\![[V\rangle]\!] &=& \mathsf{Info}(V) \in \mathbf{U}\mathbb{R}^\infty \\ [\![[M]]\!] &=& \Theta(M)\colon \mathbf{U}\mathbb{R}^\infty \to \mathbf{U}\mathbb{R}^\infty \\ [\![[T]]\!] &=& \Theta(T)\colon \mathbf{U}\mathbb{R}^\infty \times \mathbf{U}\mathbb{R}^\infty \to \mathbf{U}\mathbb{R}^\infty \end{array}$$

and similarly for the other cases. One has to take care in the case of lft's containing $\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)$ columns. So $[\![[\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\rangle]\!] = \mathbb{R}^\infty = \perp_{\mathbf{U}\mathbb{R}^\infty}$ whereas $\langle\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\rangle$ is of type real^+ and thus $[\![\langle\left(\begin{smallmatrix}0\\0\end{smallmatrix}\right)\rangle]\!] = [0,\infty] = \perp_{\mathbf{U}[0,\infty]}$. Similarly for the zero matrix $\left(\begin{smallmatrix}0&0\\0&0\end{smallmatrix}\right)$ which is interpreted as the constant-$\perp$ function of the appropriate type.

The semantics of rif-statements is given by

$$[\![\mathsf{rif}(M,N,P,Q)]\!](x) = \begin{cases} [\![P]\!](x_M) & \text{if } x_N = \emptyset \\ [\![Q]\!](x_N) & \text{if } x_M = \emptyset \\ [\![P]\!](x_M) \cup [\![Q]\!](x_N) & \text{otherwise,} \end{cases}$$

where $x_M = M^{-1}x \cap [0,\infty]$ and $x_N = N^{-1}x \cap [0,\infty]$. Hence this programming construct, introduced to enable the programmer to give definitions by cases as in the example in 3.3, may be used to produce non-single-valued functions as in the following example.

Define

$$P = \mathsf{rif}\left(\begin{pmatrix}1&0\\0&1\end{pmatrix}, \begin{pmatrix}2&-1\\1&-1\end{pmatrix}, \begin{pmatrix}0\\1\end{pmatrix}, \begin{pmatrix}1\\1\end{pmatrix}\right).$$

As $\mathsf{Info}\left(\begin{smallmatrix}1&0\\0&1\end{smallmatrix}\right) = [0,\infty]$ and $\mathsf{Info}\left(\begin{smallmatrix}2&-1\\1&-1\end{smallmatrix}\right) = [2,1]$, the restriction of $[\![P]\!]$ to maximal elements of $\mathbf{U}\mathbb{R}^\infty$, i.e. to singletons $\{x\}$ with $x \in \mathbb{R}^\infty$ is the function

$$[\![P]\!](\{x\}) = \begin{cases} \{0\} & x \in [1,2] \setminus \{1,2\} \\ \{1\} & x \in [\infty,0] \setminus \{0,\infty\} \\ \{0,1\} & x \in [0,1] \cup [2,\infty] \end{cases}$$

The deterministic reduction rules will choose one branch when evaluating such a function, where the choice depends on the particular representation of the argument.

We say that a term is *single-valued*, if such a thing does not happen, i.e. if for all its subterms of the form $\mathsf{rif}(M,N,P,Q)$ it is true that $[\![P]\!](M^{-1}x) = [\![Q]\!](N^{-1}x)$ holds for all maximal $x \in \mathsf{Info}(M) \cap \mathsf{Info}(N)$. There is no way to syntactically prevent the user from writing multi-valued programs.

Note that, as in the above example, the union of intervals $[\![P]\!](x_M) \cup [\![Q]\!](x_N)$ in definition of the semantics of rif need not be an interval. Hence it is not possible to define a denotational semantics for LCRF using the interval domain $\mathbf{I}\mathbb{R}^\infty$ in place of $\mathbf{U}\mathbb{R}^\infty$.

5 Computational Adequacy

In this paper, we have not allowed multi-valued reduction rules. This means that the reduction rules are sound, i.e. $P \to Q$ implies $[\![P]\!] = [\![Q]\!]$, except for the rif-rules. So we get

Theorem 6 (Soundness) *For all single-valued programs P,*

$$\bigsqcup_{n\in\mathbb{N}} \mathsf{Eval}(P)(n) \sqsubseteq [\![P]\!].$$

Completeness holds for a class of basic functions which we will define below. As usual, its proof is by induction. We need the following definitions.

Definition 7 *The* computable *terms of LCRF form the least set of terms such that*

- *A program P is computable whenever $[\![P]\!] \sqsubseteq \bigsqcup_{n\in\mathbb{N}} \mathsf{Eval}(P)(n)$.*
- *If $\vdash P\colon s \to t$ then P is computable whenever PQ is computable for any closed computable term Q of type s.*
- *If $x_1\colon t_1, \ldots, x_n\colon t_n \vdash P\colon t$ then P is computable whenever $[P_1, \ldots, P_n/x_1, \ldots, x_n]P$ is computable for any set of closed computable terms P_i such that $\vdash P_i\colon x_i$.*

Definition 8 *The* $\perp$-total *elements of the semantic domains $[\![t]\!]$ are defined inductively by*

- *If t is a ground type then $a \in [\![t]\!]$ is $\perp$-total iff $a = \perp$ or a is maximal.*
- *If $t = t_1 \times t_2$ then $a = (a_1, a_2) \in [\![t]\!]$ is $\perp$-total if a_1 and a_2 are $\perp$-total.*

- *If $t = t_1 \to t_2$ then $f \in [\![t]\!]$ is $\bot$-total if $f(a)$ is $\bot$-total for all $\bot$-total $a \in [\![t_1]\!]$.*

Note that every element of $\mathbb{N}_\bot$ and $\mathbb{B}_\bot$ is $\bot$-total, hence so are all elements of $[\![t]\!]$ for t a type not containing real *or* real^+*, i.e., a type of PCF with products.*

The $\bot$-total computable *terms of LCRF form the least set of terms such that*

- *A program P is $\bot$-total computable whenever $[\![P]\!]$ is $\bot$-total and $[\![P]\!] \sqsubseteq \bigsqcup_{n\in\mathbb{N}} \mathsf{Eval}(P)(n)$.*
- *If $\vdash P\colon s \to t$ then P is $\bot$-total computable whenever PQ is $\bot$-total computable for any closed $\bot$-total computable term Q of type s.*
- *If $x_1\colon t_1, \ldots, x_n\colon t_n \vdash P\colon t$ then P is $\bot$-total computable whenever $[P_1, \ldots, P_n/x_1, \ldots, x_n]P$ is $\bot$-total computable for any set of closed $\bot$-total computable terms P_i such that $\vdash P_i\colon x_i$.*

Lemma 9 *A real program P is computable ($\bot$-total computable) iff for all $a \ll [\![P]\!]$ there is $n \in \mathbb{N}$ such that $a \sqsubseteq \mathsf{Eval}(P)(n)$ (and P is $\bot$-total).*

Lemma 10 *If T is a tensor and P, Q are terms of appropriate type then there are $i, k \in \mathbb{N}$ such that for all $n \in \mathbb{N}$*

(1) $\mathsf{Eval}(\langle T\rangle(P,Q))(n+1)) = T(\mathsf{Eval}(P)(n), \mathsf{Eval}(Q(n)))$

(2) $\mathsf{Eval}(\rangle T\rangle(P,Q))(n)) = T(\mathsf{Eval}(P)(n), \mathsf{Eval}(Q(n)))$

(3) $\mathsf{Eval}([T\rangle(P,Q))(n+1)) = T(\mathsf{Eval}(P)(n+i), \mathsf{Eval}(Q(n+i)))$

(4) $\mathsf{Eval}([T](P,Q))(n+1)) = T(\mathsf{Eval}(P)(n+k), \mathsf{Eval}(Q(n+k)))$.

Proof. (1) If $P \to \langle M\rangle P'$ and $Q \to \langle N\rangle Q'$ then $\langle T\rangle PQ \Longrightarrow \langle T^{\mathsf{hd}}\rangle\langle S^{\mathsf{hd}}\rangle\ \rangle S^{\mathsf{tl}}\rangle(P', Q')$ with $S = T^{\mathsf{tl}} \textcircled{L}\, M \textcircled{R}\, N$. Thus $\mathsf{Eval}(\langle T\rangle(P,Q))(1) = \mathsf{Info}(T^{\mathsf{hd}} S^{\mathsf{hd}}) = \mathsf{Info}(T \textcircled{L}\, M \textcircled{R}\, N) = T(\mathsf{Eval}(P)(0), \mathsf{Eval}(Q)(0))$. The general case follows by induction.

Similarly, (2), (3) and (4) are proven with the alteration that in (2) the tensor does not emit prior to absorbing and that in (3) and (4) the tensor has to absorb a certain number of matrices before it can start to emit. If the arguments P and Q cannot deliver enough information for T to start absorbing, both sides of the equation are equal to $\bot$. ∎

Lemma 11 *All transformations are computable.*

Proof. Let us consider the case $\langle T\rangle$ for a tensor T. We apply Lemma 9. Suppose $a \ll [\![\langle T\rangle(P,Q)]\!] = T([\![P]\!], [\![Q]\!])$ for computable P and Q. By continuity, there are $b, c \in \mathbf{U}\mathbb{R}^\infty$ with $b \ll [\![P]\!]$, $c \ll [\![Q]\!]$, and $a \sqsubseteq T(b,c)$. By computability of P and Q there are $m, n \in \mathbb{N}$ with $b \sqsubseteq \mathsf{Eval}(P)(m)$ and $c \sqsubseteq \mathsf{Eval}(Q)(n)$. Take $k = 1 + \max(n,m)$, then $a \sqsubseteq \mathsf{Eval}(\langle T\rangle(P,Q))(k)$ by Lemma 10 and monotonicity of T. The cases of $\rangle T\rangle$, $[T]$ and $[T\rangle$, matrices and vectors are dealt with similarly. ∎

Lemma 12 *Suppose that $W = \mathsf{rif}(M,N,P,Q)$ is well-typed and single-valued, that $[\![W]\!]$ is strict, and that P and Q are $\bot$-total computable. Then W is $\bot$-total computable.*

Proof. Assume that R is $\bot$-total computable. We have to show that $W(R)$ is $\bot$-total computable. Suppose $a \ll [\![W(R)]\!]$. As R is $\bot$-total, either $[\![R]\!] = \bot$ or else $[\![R]\!]$ is maximal in $\mathbf{U}\mathbb{R}^\infty$. If $[\![R]\!] = \bot$ then $[\![W(R)]\!] = \bot$ by strictness, hence $a = \bot$ and nothing is to show. So assume that $[\![R]\!]$ is maximal. Then there is $n \in \mathbb{N}$ such that $\mathsf{Eval}(R)(n)$ is contained in $\mathsf{Info}(M)$ or $\mathsf{Info}(N)$ so that the term $W(R)$ reduces to the respective branch. Assume the former. More precisely, let $R \Longrightarrow [M_0\rangle\langle M_1\rangle \cdots \langle M_n\rangle R'$ and for $M' = M_0 \cdots M_{n-1}$ and $M'' = M' M_n$ suppose $M^* M'' \in \mathbb{M}^+$, $M^* M' \notin \mathbb{M}^+$, and $N^* M' \notin \mathbb{M}^+$. Then $W(R) \to P([M^* M''\rangle R')$ and moreover $[\![W(R)]\!] = [\![P([M^* M''\rangle R')]\!]$ by single-valuedness. Using Lemma 9, it is easy to see that $\bot$-total computability of R implies $\bot$-total computability of $[M^* M''\rangle R'$. Thus $P([M^* M''\rangle R')$ is $\bot$-total computable and hence there is $k \in \mathbb{N}$ such that $a \sqsubseteq \mathsf{Eval}(P([M^* M''\rangle R'))(k) = \mathsf{Eval}(W(R))(k)$. Therefore $W(R)$ is $\bot$-total computable. ∎

Definition 13 *A term P of type t is* basic, *if $[\![Q]\!]$ is strict, single-valued, and $\bot$-total for all subterms Q of P.*

This class includes in particular the programs for all single-valued elementary functions presented in [13].

Lemma 14 *Every term of LCRF which does not contain* rif *is computable. Every basic term is $\bot$-total computable.*

Proof. The proof is by structural induction on the term P. Let σ be a substitution of closed ($\bot$-total) computable terms for the free variables in P. Thus, we must show that σP is ($\bot$-total) computable. We only deal with the cases which deviate from the corresponding proof for PCF with products [8].

- For $P \equiv \lambda x : t.Q$, we must show that $R \equiv (\lambda x : t.\sigma Q)P_1 P_2 \ldots P_n$ is ($\bot$-total) computable if $P_1, P_2, \ldots, P_n$ are closed ($\bot$-total) computable terms and R has type real or real^+. Observe that $R \to ([P_1/x](\sigma Q))P_2 \ldots P_n \equiv ((\sigma[P_1/x])Q)P_2 \ldots P_n$, which we will call S. But, S is ($\bot$-total) computable because Q is ($\bot$-total) computable. Now $[\![R]\!] = [\![S]\!]$ and $\bigsqcup_{n\in\mathbb{N}} \mathsf{Eval}(R)(n) = \bigsqcup_{n\in\mathbb{N}} \mathsf{Eval}(S)(n)$, thus R is ($\bot$-total) computable.
- For $P \equiv \mu x : t.Q$, we must show that $R \equiv S P_1 P_2 \ldots P_n$ where $S \equiv \mu x : t.\sigma Q$ is ($\bot$-total) computable if $P_1, P_2, \ldots, P_n$ are closed ($\bot$-total) computable terms and R has type real or real^+. Define S^i by
$$\begin{aligned} S^0 &\equiv \mu x : t.x \\ S^{i+1} &\equiv (\lambda x : t.\sigma Q)\, S^i. \end{aligned}$$
Then $[\![S]\!] = \bigsqcup_{i=0}^\infty [\![S^i]\!]$ and S^i is ($\bot$-total) computable for all $i \in \mathbb{N}$. Let $a \ll [\![R]\!] = [\![S]\!][\![P_1]\!] \cdots [\![P_n]\!] = \bigsqcup_{i=0}^\infty [\![S^i P_1 \cdots P_n]\!]$. Then there is $i \in \mathbb{N}$ with $a \ll [\![S^i P_1 \cdots P_n]\!]$. Therefore there is $k \in \mathbb{N}$ such that $a \sqsubseteq \mathsf{Eval}(S^i P_1 \cdots P_n)(k)$ because S^i is ($\bot$-total) computable. In analogy with the Unwinding Theorem [8], it can be proved by induction that there is $\ell \in \mathbb{N}$ with $\mathsf{Eval}(S^i P_1 \cdots P_n)(k) \sqsubseteq \mathsf{Eval}(R)(\ell)$. Thus R is ($\bot$-total) computable.
- The remaining cases are rif, which is $\bot$-total computable by Lemma 12 and transformations, which are computable by Lemma 11. ∎

Theorem 15 (Completeness) *If P is a program which does not contain* rif *or which is basic, then*

$$[\![P]\!] \sqsubseteq \bigsqcup_{n\in\mathbb{N}} \mathsf{Eval}(P)(n).$$

Combining Theorems 6 and 15, we get computational adequacy for all programs which do not contain rif or which are basic.

Corollary 16 *The LCRF function* tan *defined in the example in Section 3.3 computes the tangent function. More precisely: If P is a program of type* real *denoting $x \in \mathbb{R}$, i.e. if $\bigsqcup_{n\in\mathbb{N}} \mathsf{Eval}(P) = \{x\}$, then* $\mathsf{tan}(P)$ *denotes* $\mathsf{tan}(x)$.

Proof. Combine the adequacy result with the proof of mathematical correctness of the formula for the tangent given in Section 3.3. ∎

Likewise, we can compute all elementary functions like sin, cos, tan, exp, ... using the formulas given in [13] and correctness of the resulting LCRF-programs is guaranteed by the adequacy result combined with the mathematical correctness of the formulas involved.

6 Conclusion

We have shown the computational adequacy of the lft framework for lazy real number computation for a class of basic real programs which includes programs for single-valued elementary functions. This in itself represents a significant theoretical progress in developing correct algorithms for real functions. It also narrows the gap between the theory and practice of real number computation. Among the main issues for future work will be the question of universality of LCRF and the development of more efficient evaluation strategies. Also, in order to capture multi-valued functions such as arctan in our semantics, we need to allow multi-valued reduction rules for rif. Alternatively, we can use a parallel construction to obtain computational adequacy for all elementary functions [14]. Furthermore, the implementation Calathea in Caml of the lft framework by Potts at Imperial College is based on *exact floating point* [15, 6], i.e. it restricts the set of matrices used to only four sign and three digit matrices; this makes the flow of information steady and allows an *information flow analysis* [13] to maximize the efficiency of the lazy evaluation. The question of computational adequacy of the exact floating point framework with the information flow analysis will also be a basic focus of future research.

References

[1] S. Abramsky and A. Jung. Domain theory. In S. Abramsky, D.M. Gabbay, and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 3, pages 1–168. Clarendon Press, 1994.

[2] J.-C. Bajard, D. Michelucci, J.-M. Moreau, and J.-M. Muller. Introduction to the special issue "Real numbers and computers". *Journal of Universal Computer Science*, 1(7):436–438, 1995.

[3] 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.

[4] P. Di Gianantonio. *A Functional Approach to Computability on Real Numbers.* PhD thesis No. TD-6/93, University of Pisa-Genova-Udine, 1993.

[5] A. Edalat. Dynamical systems, measures and fractals via domain theory. *Information and Computation*, 120(1):32–48, 1995.

[6] A. Edalat and P. J. Potts. A new representation for exact real numbers. *Electronic Notes in Theoretical Computer Science*, 6, 1997. URL: `http://www.elsevier.nl/locate/entcs/volume6.html`.

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

[8] C. A. Gunter. *Semantics of Programming Languages.* MIT Press, 1992.

[9] J. H. Lambert. *Beiträge zum Gebrauch der Mathematik und deren Anwendung*, volume 1 of *r. II.* Zweiten Theil, Berlin, 1770.

[10] D. R. Lester. Vuillemin's exact real arithmetic. In R. Heldal, C. K. Holst, and P. L. Wadler, editors, *Functional Programming, Glasgow 1991: Proceedings of the 1991 Workshop, Portree, UK*, pages 225–238, Berlin, 1992. Springer Verlag.

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

[12] G. D. Plotkin. LCF considered as a programming language. *Theoretical Computer Science*, 5:223–255, 1977.

[13] P. J. Potts. Efficient on-line computation of real functions using exact floating point. Submitted for publication. Available from: `http://theory.doc.ic.ac.uk/~pjp`, 1997.

[14] P. J. Potts. Exact real arithmetic using Möbius transformations. PhD-thesis, Imperial College London, 1998. Available from: `http://theory.doc.ic.ac.uk/~pjp`

[15] P. J. Potts and A. Edalat. Exact Real Computer Arithmetic, March 1997. Department of Computing Technical Report DOC 97/9, Imperial College, available from `http://theory.doc.ic.ac.uk/~pjp`.

[16] P. J. Potts, A. Edalat, and M. H. Escardó. Semantics of exact real arithmetic. In *Proceedings of the Twelfth Annual IEEE Symposium on Logic In Computer Science*, pages 248–257, Warsaw, Poland, 1997.

[17] D. S. Scott. Outline of a mathematical theory of computation. In *4th Annual Princeton Conference on Information Sciences and Systems*, pages 169–176, 1970.

[18] D. S. Scott. A type-theoretical alternative to ISWIM, CUCH, OWHY. *Theoretical Computer Science*, 121:411–440, 1993. Reprint of a manuscript written in 1969.

[19] M.B. Smyth. Powerdomains and predicate transformers: a topological view. In J. Diaz, editor, *Automata, Languages and Programming*, pages 662–675, Berlin, 1983. Springer Verlag. Lecture Notes in Computer Science Vol. 154.

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

[21] H.S. Wall. *Continued Fractions.* D. Van Nostrand, New York, 1948.