Refinement of rational end-points real numbers by means of floating-point numbers

Yamine Ait-Ameur*

* ENSMA-LISI, BP 109, site du Futuroscope 86960 Futuroscope Cedex, France

Communicated by M. Sintzoff; received 21 April 1997; received in revised form 4 March 1998; accepted 9 March 1998

Abstract

This paper addresses the topic of the refinement of exact real numbers. It presents a three-steps formal development towards the implementation of exact real numbers. It considers real numbers as intervals whose end-points are rational numbers. We investigate the possibility to represent these intervals by floating-point numbers as end-points in order to increase the efficiency of the implementation and to use the hardware resources. We show on an extension of the PCF language that this result can be carried out but by losing the adequacy property as defined in (Escardo, 1996). However, we show that it is possible to introduce a weak version of the adequacy property described by a Galois connection defining an abstract interpretation. Soundness and completeness properties are proved in this context. Accuracy analysis by a program analysis of the representation allows to choose between different representations of real numbers. © 1999 Elsevier Science B.V. All rights reserved.

Keywords: Exact real number computation; Interval analysis; Floating-point arithmetic; Denotational semantics; Soundness; Completeness; Accuracy analysis

1. Introduction

Several approaches for the implementation of real numbers have been addressed in the scientific computing area. The most known implementation is the one based on floating-point numbers. Although this implementation is efficient from the point of view of time consumption, it has many inconveniences, among them, e.g. cancellation, overflow, underflow and loss of precision.

On the other hand, the type Real is usually used in formal specification areas and the abstraction power of the formal specification techniques often forgets to deal with the
implementation of real numbers in full detail. The current challenge in the numerical computation area is the implementation of exact real numbers as they are used in mathematics.

This paper deals with the refinement of exact real numbers and their related computable functions and shows how such an implementation can be derived. One of the possible implementations for real numbers, issued from mathematics, is the one based on intervals where a real number is represented by an interval of rational numbers. These two rational numbers (the end-points of the interval) are defined by sequences which converge to the represented real number. Obviously, the implementation of these two sequences would not terminate since they are, in general, of infinite length.

Proceeding in this way, several approaches for the implementation of real numbers have been developed so far [3, 7, 8, 16, 18, 19, 22]. Such implementations are known as exact real number computations. They are based on lazy implementations with lists.

Refs. [3, 18, 22] gave representations of real numbers together with algorithms to compute real number functions. Refs. [7, 8] presented an extension of the PCF language [15] in order to handle real numbers. They both give a denotational semantics for this extension. Moreover, [7] gives an operational semantics and proves that Real PCF (the name of the extended language) is adequate, i.e., sound and complete.

Theoretically, the functions written in Real PCF may not terminate since they implement exact real number computations. To ensure termination, one of the possibilities is to stop the computation of the function when a given precision of the result is reached. Therefore, solutions based on the description of a precision for these sequences have been suggested [1].

Practically, this implementation:
- Is based on the approximation of a real number by a sequence of intervals (with rationals as end-points) that become smaller and smaller while the sequences converge to the represented real number.
- And by considering a given precision which allows us to stop the computations when it is reached. Implementations based on the description of a precision for these sequences have been suggested [1, 2, 12, 13]. They perform a lazy evaluation and have a parametrised precision.

We did not follow this approach although it is a promising one. We have given priority to the speed of the computation by considering intervals with floating-point numbers as end-points [1, 2]. Indeed, by implementing real numbers by intervals with floating-point numbers as end-points, the hardware implementations of arithmetic operations are used. However, the precision and the accuracy of the computations are fixed to one of the floating-point numbers. There is no possibility to make this accuracy variable (it is constant for a representation).

So, the goal of this paper is to present an implementation of the real numbers based on floating-point numbers.

It starts from the definition of real numbers based on the work of Escardo [7] where real numbers are defined by intervals with rationals as end-points. In that paper [7],
an implementation of these numbers in the PCF language is presented. An adequacy property of the extension of PCF to real numbers is presented as well.

This representation is based on lists and uses concatenations of numbers to get more precise intervals. It is based on the definition of a monoid of continuous words that represent real numbers. In the present paper, we define an implementation of the real numbers based on floating-point numbers which uses hardware machine operations. The idea consists in producing intervals for partial real numbers which are represented with floating-point numbers as end-points.

At this level, the adequacy property (i.e. soundness and completeness) as defined in [7] is lost. This is due to the fact that the mantissas and the exponents of floating-point numbers are of finite length, i.e., the length of intervals with floating-point numbers as end-points have a fixed lower bound due to the fact that floating-point numbers are finite.

However, we have defined a weak version of the adequacy property, owing to a Galois connection defining an abstract interpretation [4]. The domain of real numbers with rationals as end-points and the one of partial real numbers with floating-point numbers as end-points are linked by a pair of adjoined functions. Soundness and completeness properties of this representation are proved in this context.

Obviously, since the number of floating-point numbers is finite, this implementation will be less accurate than the one based on rational numbers and the following points are investigated:

(i) The implementation of real numbers based on intervals with rationals as end-points by intervals with floating-point numbers as end-points.

(ii) The correctness of this implementation. We must ensure that the computed interval with rational numbers as end-points is always included in the corresponding interval with floating-point numbers as end-points, in other words, the floating-point implementation is sound.

(iii) The translation of each operation on real numbers with rationals as end-points to an operation on real numbers with floating-point numbers as end-points is possible.

This paper is structured as follows. The next section recalls some definitions of computable real numbers. Section 3 is an overview of the PCF language extended with real numbers. The floating-point numbers arithmetic is introduced in Section 4. Sections 5 and 6 develop the refinement of real numbers with rationals as end-points to real numbers with floating-point numbers as end-points. Section 7 shows that this extension of the language enjoys the adequacy property with respect to the defined Galois connection. Last, we give our conclusions and the work to be performed in the future.

2. Real numbers as intervals of rationals

A real number $x$ considered as a computable real number is the limit of a computable sequence of intervals [14] with rational numbers as end points. Then, the problem of the computability of the number $x$ is the choice of the rationals and the convergence
of the sequence. Among the possible representations of computable real numbers, we can enumerate:

(i) \( x \) is represented by a computable Cauchy sequence of rational numbers \([a_0, a_1, \ldots, a_i, \ldots]\) such that the limit \( \lim_{i \to \infty} a_i = x \) and by a computable function \( q : \mathbb{N} \to \mathbb{N} \) defining the convergence rate such that \( \forall i, j, k \ |a_{q(i)+j} - a_{q(i)+k}| \leq 2^{-i} \) or with a fixed rate of convergence \( |a_i - a_{i+j}| \leq 1/i \).

(ii) \( x \) is represented by \( p \)-adic numbers as the sequence \([z_0, z_1, \ldots, z_i, \ldots]\) for every \( p \in \mathbb{N}, \ p > 1, \ \forall i, |p \times z_i - z_{i+1}| < p \) and \( x = \lim_{i \to \infty} z_i/p^i \).

(iii) Given a natural number \( b > 1 \), a negative-digit representation of \( x \) with base \( b \) is given by the sequence of integers \([z_0, z_1, \ldots, z_i, \ldots]\) such that \( \forall i \in \mathbb{N}^+, \ -b < z_i < b \) and \( x = \sum_{i \in \mathbb{N}} z_i b^{-i} \).

(iv) \( x \) is represented by the sequence of integers \([z_0, z_1, \ldots, z_i, \ldots]\) defining a continued fraction such that

\[
x = \lim_{i \to \infty} z_0 + \frac{1}{z_1 + \frac{1}{z_2 + \frac{1}{\ddots + \frac{1}{z_i}}}}
\]

For all these representations, several results and computation algorithms have been developed. They allow the computation of the sequences representing real numbers and the corresponding arithmetic operations and functions on real numbers. Naturally, the difference between the representations enumerated before is related to the efficiency of the algorithms and to the size of the integers which are obtained in each sequence [3, 7, 8, 16, 18, 19, 22]. For the purpose of our paper, we have chosen Real PCF, because it has a denotational and an operational semantics, and several proved results like adequacy [7].

3. Real PCF and partial real numbers

This section recalls the basic notions on the representation of real numbers by intervals. It uses the results from [7]. The notations are kept the same in order to have a uniform representation. For illustrating this approach on a programming language, the language PCF [15] has been chosen. The following points give a summary of these results. The complete presentation can be found in [7]. The proofs of the propositions and theorems are given in this paper as well.

3.1. The domain of partial real numbers with rationals as end-points

For simplicity and without loss of generality, only the real numbers of the interval \([0, 1]\) are considered. Let \( I \) denote the set of non-empty intervals of \([0, 1]\) whose information order is given by \( x \subseteq y \) if and only if \( y \subseteq x \). \([0, 1]\) is the bottom element with respect to this order relation. The obtained domain \( I \) is the Scott domain of partial real numbers [14, 17]. We extend \( I \) to \( I^\uparrow \) with the \( \top \) element such that \( \forall x \in I^\uparrow, x \subseteq \top \).
We denote \( x_\perp \) and \( \bar{x} \) as the left and right end-points of the partial real number \( x = [x_\perp, \bar{x}] \). For overlapping intervals, the join operator \( x \cup y = [\max(x_\perp, y_\perp), \min(\bar{x}, \bar{y})] \) and the meet operator \( x \cap y = [\min(x_\perp, y_\perp), \max(\bar{x}, \bar{y})] \) are defined on \( I \). Note that \( x \cup \top = \top \) and \( x \cap \top = x \). The order relation \( x < y \) is ensured if and only if \( \bar{x} < y \). \( x \) and \( y \) are said to be consistent, denoted \( x \simeq y \) if \( x \) and \( y \) have a common upper bound.

### 3.2. Continuous words

Let \( \Sigma \) be an alphabet, then \( \Sigma^* \) and \( \Sigma^\infty \) denote the set of finite and infinite words obtained by concatenation. An element of \( \Sigma^\infty \) is maximal iff it is an infinite word and it is finite iff it is a finite word.

In the monoid \((M, \cdot, e)\), where \( \cdot \) denotes the concatenation operator, a prefix order is defined as \( x \preceq z \) if and only if \( xy = z \) for some \( y \). The neutral element \( e \) satisfies \( xe = ex = x \).

The set of words \( \Sigma^\infty \) obtained by the concatenation \( \cdot \) together with the prefix order is a Scott domain.

Let us now come back to the partial real numbers. The following binary operation \( (x, y) \mapsto xy \) on \( I \) defined by

\[
xy = [(\bar{x} - x)y + x_\perp, (\bar{x} - x)y + \bar{x}]
\]

defines the concatenation of partial real numbers on \( I \). From a computational point of view, the concatenation \( xy \) refines the information provided by the interval \( x \). The concatenation \( xy \) selects a subinterval of \( x \) since \( x \subseteq xy \).

The left translations \( \text{cons}_a : x \mapsto ax = [(\bar{a} - a)x + a, (\bar{a} - a)x + a] \) are continuous maps for all words \( a \). Left translations are particular cases of concatenations. We denote the unique \( y \) such that \( xy = z \) by \( z/x \).

Let us recall some basic results of Escardo [7] regarding the construction of partial real number with rational end-points and the link to the continuous words.

**Theorem 3.1.** \( (I, \cdot, \perp) \) is a monoid where its prefix order coincides with the information order of \( I \).

**Theorem 3.2.** For every finite \( n \)-letter alphabet \( \Sigma \), the monoid \( \Sigma^\infty \) is isomorphic to a submonoid of \( I \).

**Theorem 3.3.** Any continuous guarded map \( f : I \rightarrow I \) has a maximal partial number as its unique fixed point.

The previous theorems allow to establish a link between the domain \((I, \subseteq)\) of partial real numbers and the monoid \((I, \cdot, \perp)\). This makes it possible to compute real numbers by performing concatenation operations, giving us an operational semantics.
3.2.1. Operations on the language

The definition of the monoid \((I, \cdot, \bot)\) and its link to the domain \(I\) have been established. It is possible to introduce operations on the words. The following operations allow to define the computations of real numbers and are defined as extensions of PCF in Real PCF.

- **Head and tails:** \(\text{tail} : \Sigma^\infty \rightarrow \Sigma^\infty\) and \(\text{head} : \Sigma^\infty \rightarrow \Sigma_\bot\) are defined as usual for lists:

\[
\begin{align*}
tail(e) &= e & \text{tail}(\sigma x) &= x \\
\text{head}(e) &= \bot & \text{head}(\sigma x) &= \sigma
\end{align*}
\]

Moreover, we define a continuous map \(\text{tail}_\alpha(x) = x/\alpha\) for every \(x\) such that \(x \leq a\) as the left inverse of \(\text{cons}_\alpha\).

- **Continuous equality:** \(\Sigma_\bot \times \Sigma_\bot \rightarrow \{tt, ff\}_\bot\) is defined by

\[
(x =_\bot y) = \begin{cases} 
  tt & \text{if } x, y \in \Sigma \text{ and } x = y \\
  \bot & \text{if } \bot \notin \{x, y\} \\
  ff & \text{if } x, y \in \Sigma \text{ and } x \neq y 
\end{cases}
\]

It gives the best result with respect to interval approximation. The undefined element is returned if one of the arguments is undefined.

- **Continuous comparison map:** \(I \times I \rightarrow \{tt, ff\}_\bot\) is defined by

\[
(x <_\bot y) = \begin{cases} 
  tt & \text{if } x < y \\
  \bot & \text{if } x \simeq y \\
  ff & \text{if } x > y 
\end{cases}
\]

The comparison map returns the boolean undefined element \((\bot)\) when the comparison cannot be asserted exactly, i.e. when \(x\) and \(y\) are consistent (they have a common upper bound).

A useful comparison operator \(\text{head}_r(x)\) is defined by \(\text{head}_r(x) = x <_\bot r\).

- **The conditional:** Having introduced equality and comparison, the conditional is defined by the parallel \(/\!f\). It has been introduced in [7]. \(p/\!f : \{tt, ff\}_\bot \times I \times I \rightarrow I\) is defined by

\[
pif p \text{ then } x \text{ else } y = \begin{cases} 
  x & \text{if } p = tt \\
  x \sqcap y & \text{if } p = \bot \\
  y & \text{if } p = ff
\end{cases}
\]

It is an extension of the classical conditional and it ensures that the best information (with respect to interval approximation) compatible with \(x\) and \(y\) is produced when \(p\) evaluates to \(\bot\).

The next theorem shows a representation of a partial real number \(x\) in Real PCF.
Theorem 3.4. Let $L = [0, 1/2]$ and $R = [1/2, 1]$. For any partial number $x \in L$,

\[ x = \text{pif} \ head_{1/2}(x) \text{ then } cons_L(tail_L(x)) \text{ else } cons_R(tail_R(x)) \]

This theorem gives a computation mechanism for partial real numbers with rational end-points. It shows how it is possible to compute a real number using the set of operators previously introduced and defined on the monoid. The operators introduced previously allow to define a complete arithmetic on partial real numbers. Indeed, definitions for addition, multiplication, average, logarithm, and so on can be found in [7]. As an example, the complement operation can be defined as follows:

\[ \text{comp}(x) = 1 - x = \text{pif} \ head_{1/2}(x) \text{ Then } cons_R(1 - tail_L(x)) \text{ Else } cons_L(1 - tail_R(x)) \]

More details on these definitions and the proofs of the theorems can be found in [7]. The complete definition of the partial real number arithmetic is given as well.

Now, the whole denotational semantics of the PCF language extended with real numbers represented as intervals with rational end-points can be presented.

3.3. Semantics of Real PCF

The syntax of the language is a classical lambda calculus. The PCF terms are described as $L :: c | x | MN | \lambda x. M$. They respectively define a constant, a variable, the application and the abstraction. The denotational semantics of the PCF language extended with real numbers is given below. It introduces domains, environments and semantic equations that give the denotation of each program construction. Each basic construction is interpreted by its corresponding function expressing its meaning.

3.3.1. Domains

Let us consider $D_\sigma$ as the domain corresponding to the type $\sigma$. \{$D_\sigma$\} is a collection of domains, one for each type $\sigma$. \{$D_\sigma$\} contains the domain $I$ of partial real numbers.

3.3.2. Interpretation and environments

An interpretation function $A$ of a language $L$ is a function $A : L \rightarrow \bigcup \{D_\sigma\}$ which gives the interpretation of the constants and of the basic functions of the language $L$.

Environments $\rho : \text{var} \rightarrow \bigcup \{D_\sigma\}$ link variables to values in the domain $D_\sigma$ corresponding to their type.

In order to give the semantics of the PCF language extended with real numbers, the following basic functions are interpreted by the function $A : L \rightarrow \bigcup \{D_\sigma\}$ where $L$ is the set of the basic language constructions. Its definition is given as

\[ A[\text{cons}_a] = \text{cons}_a \]

\[ A[\text{tail}_a] = \text{tail}_a \]
\[ \mathcal{A}[\text{head}] = \text{head}, \]
\[ \mathcal{A}[\text{pif}]_{p, x, y} = \text{pif}(p, x, y) \]

The basic functions of the Real PCF language are interpreted by the functions of the monoid \( (I, \cdot, \bot) \) defined in Section 3.2.1.

### 3.3.3. The denotational semantics

The denotational semantics is given by the semantic function \( \mathcal{A} : \text{Terms} \rightarrow \text{Env} \rightarrow \bigcup \{ D_\sigma \} \) is defined inductively on the structure of the terms by

\[ \mathcal{A}[x]_p = \rho(x) \]
\[ \mathcal{A}[c]_p = \mathcal{A}[c] \]
\[ \mathcal{A}[MN]_p = \mathcal{A}[M]_p(\mathcal{A}[M]_p) \]
\[ \mathcal{A}[^\lambda x.M]_{p,x} = \mathcal{A}[M]_{p[x/x]} \text{ with } x \in D_\sigma \text{ if } x : \sigma \]

### 3.4. Operational semantics

One of the major interest of the Real PCF language is the existence of an operational semantics as well. It is not needed for the purpose of our paper, but we introduce it in order to be able to set clearly the adequacy property.

A set of reduction rules defining the operational semantics of real PCF are defined. Each reduction rule is defined as \( M \rightarrowtail M' \) where \( M \) and \( M' \) are real PCF terms.

The successive application of reduction operation has been defined as the \( \text{Eval} \) operator defined as \( \text{Eval}(M) = c \) if and only if \( M \rightarrowtail c \) where \( c \) is a term that cannot be reduced.

Escardo [7] has proved that real PCF satisfies the adequacy property which can be stated by the following theorem.

**Theorem 3.5.**

\[ \text{Eval}(M) = c \iff [M] = c \]

We can unfold this property as the two following properties:

- **Soundness** \( \subseteq \text{Eval}(M) \subseteq \mathcal{A}[M] \)
- **Completeness** \( \supseteq \text{Eval}(M) \supseteq \mathcal{A}[M] \)

The proof of this theorem can be found in [7].

The rest of the paper is devoted to the definition of another implementation of real numbers as intervals with floating-point numbers as end-points. Instead of having infinite sequences of intervals denoting a real number, we will focus on a finite sequence with a fixed precision, i.e., intervals with floating point as end-points. This approach gives a computable approximation of the previous computations which is based on the efficient operations of the floating-point numbers.
Note that due to the finite number of floating-point numbers, the sequences coding partial real numbers will be finite, and therefore the maximum precision we can get is the length of the interval with two consecutive floating-point numbers as end-points. These notions are introduced below.

4. Floating-point numbers

This section introduces the basic concepts of floating-point numbers following the descriptions of [9, 11]. For the purpose of this paper, we will consider only the floating-point numbers belonging to the unit interval \([0, 1]\). The interval is named the unit floating-point interval and is denoted by \(F_1\).

**Definition 4.1.** According to Knuth [11], a floating-point number is represented by a pair \((m, e)\) where \(m\) and \(e\) are integer numbers. For a base \(B\) and an excess code \(q\), the floating-point number \((m, e)\) represents the real number \(m \times B^{e-q}\). \(m\) and \(e\) are respectively named the mantissa and the exponent of the represented floating-point number. In order to get a canonical representation of a given number, each of these numbers is normalised, i.e. the most significant digit is not equal to zero.

4.1. Floating-point arithmetic

Arithmetic operations have been defined on these numbers and are mainly supported by the hardware. The following basic arithmetic operations are implemented for floating-point numbers.

(i) **Addition and subtraction:** Let us consider that \(e_e \leq e_u\). The addition \((m_w, e_w) = (m_u, e_u) \oplus (m_v, e_v)\) is defined by \(m_w = m_u + (m_v/b^{e_u-e_v})\) and \(e_w = e_u\). The exponent is the one of \(u\) and the mantissas are added after equalising the exponents (alignment). The division \(m_v/(b^{e_u-e_v})\) corresponds to a shift of the mantissa of \(e_u - e_v\) digits to ensure the equality of the exponents.

(ii) **Multiplication:** \((m_w, e_w) = (m_u, e_u) \otimes (m_v, e_v)\) is given by \(m_w = m_u m_v\) and \(e_w = e_u + e_v - q\). The two mantissas are multiplied and the exponents are added.

(iii) **Division:** The division is defined by the inverse operator. The result of \((m_w, e_w) = (m_u, e_u) \odot (m_v, e_v)\) is given by \(m_w = (1/b) m_u/m_v\) and \(e_w = e_u - e_v + q + 1\). The exponents are subtracted and the mantissas are divided.

4.2. Basic definitions

4.2.1. **Length of the mantissa**

The length of a mantissa is the number of digits needed to represent a mantissa in a base \(B\). For a floating-point number \(x\) the size of its mantissa is denoted by \(S_x\). The maximum authorised number of digits is denoted \(S_m\) and the maximum value of the exponent in length is written as \(V_e\). It is fixed by the machine support or by
the compiler. The definition of the values $S_m$ and $V_e$ makes the set of floating-point numbers a finite set.

4.2.2. Consecutive floating-point numbers

Two floating-point numbers $(m_a, e_a)$ and $(m_b, e_b)$ belonging to $F_I$ are consecutive if and only if one of the following conditions hold:

(i) if $e_a = e_b$ then $m_b = m_a + 0.1 \times B^{-e_a - S_i}$
(ii) if $e_a = e_b - 1$ then all the digits of $m_a$ are equal to $B - 1$ and $m_b = 0.1$
(iii) if one of the numbers is 0, then 0 and the least floating-point number representable in the unit interval are consecutive floating-point numbers,
(iv) if one of the numbers is the biggest representable floating-point number in the unit interval less than 1, then this number and 1 are consecutive floating-point numbers.

4.2.3. Maximal precision

For a number $x \in [x_F, \bar{x}_F]$, where $x_F$ and $\bar{x}_F$ are floating-point numbers, the maximal precision for $x$ is reached when $x_F$ and $\bar{x}_F$ are consecutive and is equal to $|x_F - \bar{x}_F|$.

4.2.4. Roundings

According to the IEEE-754 [10] standard, we extend the rounding method to the unit interval $F_I$. In case of overflows or cancellations, the nearest floating-point number is produced and all the values produced by floating-point operations that are:

- less than the consecutive floating-point number of 0, are rounded to 0,
- greater than the floating-point number whose consecutive is 1, are rounded to 1.

This notion of roundings will be extended later in this paper to the roundings of intervals with floating-point numbers as end-points.

5. Refinement of real numbers domain by floating-point domain

The finite domain of intervals with floating-point numbers as end-points is introduced to implement real number arithmetic and rational approximations.

In order to define a refinement operation from the domain $I$ to the domain $I_F$ of real numbers with floating-point numbers as end points, we have to introduce the structure of this domain, and the related operations. Moreover, we will indicate how the different orders defined on $I$ map on the domain $I_F$ and finally the embedding of the operations on $I$ to $I_F$.

First, we will define the domain $I_F$ and the corresponding orders and operations. Second, the domain $I$ is refined to what we named arbitrary floating-point numbers. Last, we give the refinement of arbitrary floating-point numbers to the elements of $I_F$. 
5.1. The domain of floating-point numbers intervals

**Definition 5.1.** The set $IF$ of non-empty subintervals of $F_i$ ordered by the reverse inclusion defined as $\forall x, y \in IF$, $x \subseteq_F y$ iff $y \subseteq x$, with the interval $F_i$ as bottom element $\bot_F$, i.e. $\forall x \in IF, F_i \subseteq_F x$ is a Scott domain. The left and right end-points of the interval $x$ will be denoted by $x_{\overline{F}}$ and $\overline{x_{F}}$, respectively. When no ambiguity occurs, they will be written as $\overline{x}$ and $\overline{y}$.

As for the domain $I$, the join and meet operators are defined. If $x$ and $y$ are two overlapping intervals of $IF$, then the join $\cup_F$ and the meet $\cap_F$ operators are defined by $x \cup_F y = [\max(x_{\overline{F}}, \min(\overline{x}, \overline{y}))]$ and $x \cap_F y = [\min(x_{\overline{F}}, \max(\overline{x}, \overline{y}))]$.

From $IF$, we build the complete lattice $IF_L$ by adding the element $\top_F$ to $IF$ such that $\forall x_F \in IF_L$, $x_F \subseteq_F \top_F$, $x_F \cap_F \top_F = x_F$ and $x_F \cup_F \top_F = \top_F$.

5.1.1. Orders on real numbers with floating-point numbers as end-points

Let us assume that $x = [\overline{x}, \overline{y}]$ and $y = [\overline{y}, \overline{y}]$ are two elements of $IF_L$, then we define

(i) $x =_F y$ iff $x = y$

(ii) $x <_F y$ iff $\overline{x} < \overline{y}$

(iii) $x >_F y$ iff $\overline{x} > \overline{y}$

(iv) Two elements $x$ and $y$ are consistent, $x \simeq_F y$, if they have a common upper bound. From these definitions, we can derive the following definitions of continuous orders in the domain $IF_L$ of real numbers with floating point numbers as end-points. They are used in the definition of Real PCF programs where real numbers are represented by floating-point numbers as end-points.

- **Continuous equality operator** $I^L \times I^L \rightarrow \{tt, ff\}_{\bot_F}$ is defined by

$$x =_{\bot_F} y = \begin{cases} 
  tt & \text{if } x =_F y \\
  \bot & \text{if } \bot_F \in \{x, y\} \\
  ff & \text{if } x \neq_F y 
\end{cases}$$

If one of the operands is undefined, then the result is also undefined.

- **Continuous comparison operator** $I^L \times I^L \rightarrow \{tt, ff\}_{\bot_F}$ is defined by

$$x <_{\bot_F} y = \begin{cases} 
  tt & \text{if } x <_F y \\
  \bot & \text{if } x \simeq_F y \\
  ff & \text{if } x >_F y 
\end{cases}$$

The comparison of $x$ and $y$ is undefined if they are consistent.

These definitions are used below to define the refinement of $I^L$ by $I_F^L$.

5.2. Real numbers as intervals with arbitrary floating-point numbers as end-points

As stated above, floating-point numbers are defined by pairs of integer numbers. Without loss of generality, we will consider the unit interval $I$ of real numbers where real numbers are represented by intervals with rational end-points.
A rational number, belonging to the set \( \mathbb{Q}_I \) (the set of rationals contained in \( I \)) is transformed to a pair of numbers defining an arbitrary floating-point number. The \textit{make} function which achieves this transformation is defined by

\[
\text{make} : \mathbb{Q}_I \rightarrow \mathbb{Q}_I \times \mathbb{N}
\]

where \( \mathbb{N} \) is the set of natural numbers.

In the same way, two functions which allow to build the mantissa and the exponent of a number with arbitrary lengths are introduced:

\[
\text{exp}: \mathbb{Q}_I \rightarrow \mathbb{N} \quad \text{and} \quad \text{mant}: \mathbb{Q}_I \rightarrow \mathbb{Q}_I
\]

They are defined by

\[
\text{exp}(x) = \begin{cases} 
1 + \text{exp}(\text{tail}(x)) & \text{if } \text{head}(x) = 0 \\
0 & \text{else}
\end{cases}
\]

and

\[
\text{mant}(x) = \begin{cases} 
\text{mant}(\text{tail}(x)) & \text{if } \text{head}(x) = 0 \\
x & \text{else}
\end{cases}
\]

The \textit{mant} and \textit{exp} functions can loop indefinitely if the rational number \( x \) is of infinite length. In practice, this function terminates when the length of the mantissa \( S_m \) is reached. A program analysis is performed at this stage in order to stop the computation of the digits when the suited precision is reached (by lazy evaluation).

Now, we can define the transformation function, which produces a number defined by a pair

\[
\text{make}(x) = (\text{mant}(x), -\text{exp}(x))
\]

The sign of the exponent will be omitted in the following. The mantissa is positive and belongs to \( \mathbb{Q}_I \).

The number \( \text{make}(x) \) is an arbitrary floating-point number.

5.3. Arbitrary floating-point numbers represented by floating-point numbers

In the previous section, a rational number was translated to an arbitrary floating-point number eventually with an infinite mantissa.

We show in the following how these arbitrary floating-point numbers are restricted to floating-point numbers of \( F_I \).

5.3.1. Building floating-point numbers from arbitrary floating-point numbers

Let us consider the function \textit{make}_\# which builds a floating-point interval representing the best approximation of an arbitrary floating-point number:

\[
\text{make}_\# : \mathbb{Q}_I \times \mathbb{N} \rightarrow (\mathbb{Q}_I \times \mathbb{N}) \times (\mathbb{Q}_I \times \mathbb{N})
\]
Defined by

\[ \text{make}_n(m, e) = ((m_r, e_r), (m_h, e_h)) \]

where \((m_r, e_r)\) and \((m_h, e_h)\) are consecutive floating-point numbers such that
- \(m_r\) and \(m_h\) are the low and high mantissas of the floating-point numbers determining the interval containing \((m, e)\). They are determined by
  \[ m_r \leq m \leq m_h \] and \(m_r\) and \(m_h\) are consecutive.
- \(e_r\) and \(e_h\) are the low and high exponents of the floating-point numbers determining the interval containing \((m, e)\). Either \(e_r = e_h\) or \(e_r = e_h - 1\).

In addition, two projection functions \(\text{get}_r : (Q_l \times N) \times (Q_l \times N) \rightarrow Q_l \times N\) and \(\text{get}_h : (Q_l \times N) \times (Q_l \times N) \rightarrow Q_l \times N\) are introduced. They are defined by

\[ \text{get}_r((m_r, e_r), (m_h, e_h)) = (m_r, e_r) \]
\[ \text{get}_h((m_r, e_r), (m_h, e_h)) = (m_h, e_h) \]

The following theorem ensures that each real number can be bounded by an interval with floating-point numbers as end-points.

**Theorem 5.1.**

\( \forall x \in [0, 1], \quad x \in \text{make}_n(\text{make}(x)) \)

**Proof.** Let us consider \(x \in [0, 1]\). Then, \(\text{make}(x) = (\text{mant}(x), -\text{exp}(x))\). It is the corresponding representation of \(x\) and possibly an infinite one. By applying, \(\text{make}_n(\text{mant}(x), -\text{exp}(x))\), and truncating the representations to mantissas of size less or equal to \(S_m\), we get a pair of consecutive floating-point numbers \(((m_r, e_r), (m_h, e_h))\) such that \((m_r, e_r) \leq (\text{mant}(x), -\text{exp}(x)) \leq (m_h, e_h)\) and then \((m_r, e_r) \leq x \leq (m_h, e_h)\). \(\square\)

5.4. Refinement of the partial real numbers domain by the floating-point domain

Now, given a rational number \(q\), we are able to get the most accurate interval which represents \(q\). Moreover, it is possible to represent in this way an approximation of the number \(x\) represented by the rational end-point interval \([\underline{x}, \bar{x}]\). The idea is to find floating-point numbers \(a, b, c, d\) such that \(a \leq \underline{x} \leq b\) and \(c \leq \bar{x} \leq d\) where \(a, b\) and \(c, d\) are respectively consecutive floating-point numbers.

**Definition 5.2.** Any rational number \(\underline{x}\) or \(\bar{x}\) (representing the partial real number \(x = [\underline{x}, \bar{x}]\)) is refined in a floating-point representation by the function \(\text{ref} : [0, 1] \rightarrow F_l\) defined:

\[ \text{ref} = \text{make}_n \circ \text{make} \]

**Corollary 5.3.**

\( \forall x \in [0, 1], \quad x \in \text{ref}(x) \)
Proof. This theorem is equivalent to Theorem 5.1. □

5.4.1. Computation of the ref function

In practice, computers do not allow the representation of such numbers. The number of digits representing the mantissa is finite and is equal to $S_m$.

The function ref defined as $\text{make}_\| \circ \text{make}$ needs to be analysed. Indeed, since the make function can loop (i.e. in case of a mantissa with infinite length), the $\text{make}_\|$ function has to be lazy evaluated. In case of mantissas whose sizes become greater than $S_m$ and when the size of the mantissa produced the make function reaches $S_m + 1$, the $\text{make}_\|$ function truncates the mantissa and returns the corresponding floating-point numbers. Therefore, several rational numbers will have a common representation.

5.4.2. Real numbers as intervals with floating-point numbers as end-points

Now, the function ref applied to the interval $[x, \bar{x}]$ gives intervals of the form

$$[\text{ref}(x), \text{ref}(\bar{x})] = [(m_l, e_l), (m_h, e_h)]$$

which are exact approximations of the interval $[x, \bar{x}]$. In order to keep the same representation as the one for rational numbers, we take, in the interval, the left projection for the low rational and the right projection for the high rational.

$$[\text{get}_l(\text{ref}(x)), \text{get}_h(\text{ref}(\bar{x}))] = [(m_l, e_l), (m_h, e_h)]$$

Informally, the number $x$ is approximated by the smallest floating point interval which approximates the interval $[x, \bar{x}]$.

5.4.3. The refinement function: definition

For the rest of the paper, we will note $\text{Ref} : I^L \rightarrow I^F$

$$\text{Ref}(x) = \text{Ref}([x, \bar{x}]) = [\text{get}_l(\text{ref}(x)), \text{get}_h(\text{ref}(\bar{x}))]$$

Note that due to Definition 5.2, $\text{Ref}(\perp) = \text{Ref}([0, 1]) = \perp_F = [0, 1]$ and $\text{Ref}(\top) = \top_F$.

5.4.4. Correctness

The correctness criteria for this representation is given by the following:

$$\forall x \in I, \ [x, \bar{x}] \subseteq [\text{get}_l(\text{ref}(x)), \text{get}_h(\text{ref}(\bar{x}))]$$

It ensures the safety of the refinement.

Theorem 5.4.

$$\forall x \in I, \ [x, \bar{x}] \subseteq [\text{get}_l(\text{ref}(x)), \text{get}_h(\text{ref}(\bar{x}))]$$

which can be written as

$$\forall x \in I, \ [x, \bar{x}] \subseteq \text{Ref}(x)$$
Proof. To prove this theorem, it is enough to prove that $\text{get}_h(\text{ref}(x)) \leq x$ and $\bar{x} \leq \text{get}_h(\text{ref}(\bar{x}))$.

From Corollary 5.4 it appears that, $x \in \text{ref}(x) = ((m_l, e_l), (m_h, e_h))$ which means that $\text{get}_h(\text{ref}(x)) = (m_l, e_l) \leq x$.

Dually, $\bar{x} \in \text{ref}(\bar{x}) = ((\bar{m}_l, \bar{e}_l), (\bar{m}_h, \bar{e}_h))$ which means that $\bar{x} \leq (\bar{m}_h, \bar{e}_h) = \text{get}_h(\text{ref}(\bar{x}))$.

Finally, we can assert that $[x, \bar{x}] \subseteq [\text{get}_h(\text{ref}(x)), \text{get}_h(\text{ref}(\bar{x}))]$.

The following theorem shows that the floating-point numbers implementation is less precise than the rational numbers representation.

Theorem 5.5.

$\exists x = [x, \bar{x}] \in I$. $\text{Ref}(x) \not\subset x$

This theorem shows that the representation of partial real numbers by floating-point numbers as end-points is not isomorphic.

Proof. By Corollary 5.5, we know that $x \in \text{ref}(x)$, so $\text{get}_h(\text{ref}(x)) \leq x \leq \text{get}_h(\text{ref}(\bar{x}))$ and $\bar{x} \in \text{ref}(\bar{x})$, so $\text{get}_h(\text{ref}(\bar{x})) \leq \bar{x} \leq \text{get}_h(\text{ref}(\bar{x}))$. So, the interval $\text{Ref}(x)$ is larger than the interval denoted by $x$, therefore there exist elements of $\text{Ref}(x)$ that are not contained in $x$. □

5.5. The representation function: definition

Let us introduce the function $\text{Rep} : I_F \rightarrow I_L$ defined by

$\forall x_F \in I_F^L$, $\text{Rep}(x_F) = x$ such that $x = x_F$ and $\bar{x} = \bar{x_F}$.

The representation function $\text{Rep}$ characterises the set of partial real numbers that can be represented by a given $x_F \in I_F^L$. We get

$\text{Rep}(\bot_F) = \bot$

$\text{Rep}(x_F) = [x_F, \bar{x_F}]$, $\forall x_F \in I_F$

$\text{Rep}(\top_F) = \top$

The following proposition enables the establishment of a sound link between the two domains $I_L$ and $I_F$ owing to a Galois connection, and therefore it defines an abstract interpretation [4–6].

Proposition 5.6. $(I_L, \text{Ref}, \text{Rep}, I_F^L)$ defines a Galois connection, i.e.

$\forall x \in I_L$, $\forall x_F \in I_F^L$, $\text{Ref}(x) \sqsubseteq_F x_F \iff x \sqsubseteq \text{Rep}(x_F)$
This proposition is equivalent to the following properties:
\( \forall x, y \in \mathcal{I}^l, \forall x_F, y_F \in \mathcal{I}_F^l, \)

(i) \( \text{Ref} \) is monotone, i.e. \( x \subseteq y \Rightarrow \text{Ref}(x) \subseteq \text{Ref}(y) \)
(ii) \( \text{Rep} \) is monotone, i.e. \( x_F \subseteq y_F \Rightarrow \text{Rep}(x_F) \subseteq \text{Rep}(y_F) \)
(iii) \( x_F \subseteq \text{Ref}(\text{Rep}(x_F)) \)
(iv) \( \text{Rep}(\text{Ref}(x)) \subseteq x \)

Proof. (i) Consider \( x \) and \( y \) in \( \mathcal{I}^l \), such that \( x \subseteq y \), then

\[
x \subseteq y \Leftrightarrow [y, \bar{y}] \subseteq [x, \bar{x}]
\]
then, \( \text{Ref}(x) = [\text{get}_r(\text{Ref}(x)), \text{get}_l(\text{Ref}(x))] \) and \( \text{Ref}(y) = [\text{get}_r(\text{Ref}(y)), \text{get}_l(\text{Ref}(y))] \). By Theorem 5.4, we obtain that \( \text{Ref}(x) \subseteq \text{Ref}(y) \).
(ii) \( \text{Rep} \) is monotone since it is defined as the identity function.
(iii) Let \( x_F \in \mathcal{I}_F^l \); then \( \text{Rep}(x_F) = [x_F, \bar{x}_F] \) by Section 5.5. Since \( \text{Ref}(x_F) = x_F \), we obtain \( x_F \subseteq \text{Ref}(\text{Rep}(x_F)) \).
(iv) Let \( x \) be in \( \mathcal{I}^l \). Since \( x \subseteq \text{Ref}(x) \), by Theorem 5.4, we have \( [\text{Ref}(x), \text{Ref}(x)] \subseteq x \), so \( \text{Rep}(\text{Ref}(x)) \subseteq x \) since \( \text{Rep} \) is the identity function. \( \square \)

Note, that the interest of the definition of the Galois connection, is that for any \( x \in \mathcal{I} \), \( \text{Ref}(x) \) represents the best approximation of \( x \) in the domain \( \mathcal{I}_F^l \). Moreover, \( \text{Ref} \) and \( \text{Rep} \) can respectively be considered as the abstraction and concretisation functions.

Remark. The \( \text{Rep} \) function has been defined as identity. In fact, it represents the set of all partial real numbers with rationals as end-points of \( \mathcal{I}^l \) that can be represented with a partial real number with floating-point numbers as end-points. It has been defined as identity for convenience and in order to simplify the proofs. Otherwise, we would have to deal with powersets [6].

5.6. Orders of \( \mathcal{I}_F^l \)

To allow the order mappings, the orders defined in the domain \( \mathcal{I}_F^l \) are related to the orders on \( \mathcal{I}^l \). The following propositions formally state these relationships.

Proposition 5.7. For any partial real numbers \( x, y \in \mathcal{I}^l \) with rational end-points, the following holds:

(i) \( \text{Ref}(x) \triangleleft^F \text{Ref}(y) \Rightarrow x < y \)
(ii) \( \text{Ref}(x) \triangleright^F \text{Ref}(y) \Rightarrow x > y \)
(iii) \( x \simeq y \Rightarrow \text{Ref}(x) \simeq^F \text{Ref}(y) \)

Proof. Let \( x, y \in \mathcal{I}_F^l \).
(i) By Theorem 5.4, we know that \( x \subseteq \text{Ref}(x) \) and \( y \subseteq \text{Ref}(y) \), so

\[
\text{Ref}(x) \triangleleft^F \text{Ref}(y) \Leftrightarrow \overline{\text{Ref}(x)} \triangleleft \overline{\text{Ref}(y)}
\]
\[ \iff \text{get}_h(\text{make}_n(\text{make}(x))) \]
\[ < \text{get}_l(\text{make}_n(\text{make}(y))) \]

and since \( \bar{x} \leq \text{get}_h(\text{make}_n(\text{make}(\bar{x}))) \) and \( \text{get}_l(\text{make}_n(\text{make}(\underline{y}))) \leq \underline{y} \), we get \( \bar{x} < \underline{y} \).

(ii) The previous reasoning applies.

(iii) \( x \preceq y \) means that \( x \) and \( y \) have a common upper bound which is either \( \bar{x} \) or \( \underline{y} \).

So, \( \text{Ref}(x) \) and \( \text{Ref}(y) \) have a common upper bound as well since \( x \subseteq \text{Ref}(x) \) and \( y \subseteq \text{Ref}(y) \).

Proposition 5.8. For any partial real numbers \( x, y \in I^L \) with rational end-points, the following holds:

\[ x = \bot y \Rightarrow \text{Ref}(x) = \bot \text{Ref}(y) \]

Proof. Assume \( x = \bot y \). If \( x = y \) then, \( \text{Ref}(x) = \text{Ref}(y) \) and when \( x = \bot \) then \( \text{Ref}(x) = \bot \text{Ref}(y) \) and so, \( \text{Ref}(x) = \bot \text{Ref}(y) \).

Proposition 5.9. For any partial real numbers \( x, y \in I^L \) with rational end-points, the following holds:

\[ \text{Ref}(x) <_{\bot, \text{Ref}} \text{Ref}(y) \Rightarrow x <_{\bot} y \]

Proof. From Proposition 5.7, we know that \( \text{Ref}(x) <^F \text{Ref}(y) \Rightarrow x < y \) and \( \text{Ref}(x) >^F \text{Ref}(y) \Rightarrow x > y \) and that \( x \preceq y \Rightarrow \text{Ref}(x) \preceq^F \text{Ref}(y) \), then from the definition of \( <_{\bot} \) we can assert that \( \text{Ref}(x) <_{\bot, \text{Ref}} \text{Ref}(y) \Rightarrow x <_{\bot} y \).

Proposition 5.10. For any partial real number with rational end-points, the following holds:

(i) \( \text{Ref}(x \sqcup y) \subseteq \text{Ref}(x) \sqcup F \text{Ref}(y) \) or \( \text{Ref}(x) \sqcup F \text{Ref}(y) \subseteq F \text{Ref}(x \sqcup y) \)

(ii) \( \text{Ref}(x \sqcap y) \subseteq \text{Ref}(x) \sqcap F \text{Ref}(y) \) or \( \text{Ref}(x) \sqcap F \text{Ref}(y) \subseteq F \text{Ref}(x \sqcap y) \)

Proof. The proof is immediate from the definitions of the join and meet operators.

(i) From the definition of \( \sqcup F \), it follows that

\[ \text{Ref}(x \sqcup y) = \text{Ref}([\text{max}(\bar{x}, \underline{y}), \text{min}(\bar{x}, \underline{y})]) \]
\[ = [\text{get}_l(\text{ref}(\text{max}(\bar{x}, \underline{y}))), \text{get}_h(\text{ref}(\text{min}(\bar{x}, \underline{y})))] \]
\[ \subseteq [\text{max}(\text{get}_l(\text{ref}(\bar{x}, \underline{y}))), \text{min}(\text{get}_h(\text{ref}(\bar{x}, \underline{y})))] \]
\[ \subseteq \text{Ref}(x) \sqcup \text{Ref}(y) \]

(ii) The same reasoning as before applies.

The next section shows how a function \( f \) on the domain \( I^L \) can be mapped to a function \( \bar{f} \) on the domain \( I^F \). We will assert the correctness condition of this mapping.
as well. The $\tilde{f}$ function is the abstract function associated to the function $f$ with respect to the Galois connection defining the abstract interpretation.

6. Mapping of the operations of the domain $I^L$ to the $I^L_F$ domain

In order to obtain the implementation of real numbers represented by intervals with rational end-points by real numbers represented by intervals with floating-point numbers as end-points, we need to:
- Define a transformation of each basic function $f$ defined for real numbers intervals with rational numbers as end-points to a function $\tilde{f}$ defined on real number intervals with floating-point numbers as end-points.
- Ensure the safety, i.e., ensure that the function $\tilde{f}$ on $I^L_F$ is a correct implementation (approximation) of $f$ on $I^L$.
- Ensure completeness, i.e. ensure that to each function $f$ on $I^L$, corresponds a function $\tilde{f}$ on $I^L_F$.

The operations defined on the domain $I^L$ are mapped to the domain $I^L_F$. The relations between the obtained operators and the original ones and the different propositions which ensure that the obtained operations on $I^L_F$ correctly refine the ones on $I^L$ are expressed in this section.

6.1. Correct approximation

For any function $f : I^L \rightarrow I^L$ and its corresponding function $\tilde{f} : I^L_F \rightarrow I^L_F$, we have, $\forall x \in I^L, f([x, \bar{x}]) \subseteq \tilde{f}([\text{get}_r(\text{ref}(x)), \text{get}_l(\text{ref}(\bar{x}))]) = \tilde{f}(\text{Ref}(x))$.

6.2. Operations on $I^L_F$

The mapping of the basic functions, on real numbers as intervals with rationals as end-points defined in [7], into the domain $I^L_F$ are defined by
- $\text{cons}_a$: the corresponding function is $\text{cons}_a : I^L_F \rightarrow I^L_F$ and is defined by
  \[ \text{cons}_a(\text{Ref}(x)) = \text{Ref}(\text{cons}_a(x)) = \text{Ref}(ax) \]
- $\text{tail}_a$: the corresponding function is $\text{tail}_a : I^L_F \rightarrow I^L_F$ and is defined by
  \[ \text{tail}_a(\text{Ref}(x)) = \text{Ref}(\text{tail}_a(x)) = \text{Ref}(x/a) \]
- $\text{head}_r$: the corresponding function is $\text{head}_r : I^L_F \rightarrow T$ and is defined by
  \[ \text{head}_r(\text{Ref}(x)) = \text{Ref}(x) < \bot, \text{Ref}(r) \]
the corresponding parallel conditional is defined by \( \tilde{p}f \) as
\[
pf(p, \text{Ref}(x), \text{Ref}(y)) = \begin{cases} \text{Ref}(x) & \text{if } p = \text{tt} \\ \text{Ref}(x) \cap_F \text{Ref}(y) & \text{if } p = \perp \\ \text{Ref}(y) & \text{if } p = \text{ff} \end{cases}
\]

Proposition 6.1.

(i) \( xy \subseteq \text{Ref}(xy) \)
(ii) \( \text{cons}_a(x) \subseteq \text{cocons}_a(\text{Ref}(x)) \)
(iii) \( \text{tail}_a(x) \subseteq \text{tail}_a(\text{Ref}(x)) \)
(iv) \( \text{head}_a(\text{Ref}(x)) \Rightarrow \text{head}_a(x) \)
(v) \( \text{pf}(p, x, y) \subseteq \text{pf}(p, \text{Ref}(x), \text{Ref}(y)) \)

Proof. The proof is immediate. It follows from the previous propositions.

(i) \( xy = [(2 - x)y + x, (2 - x)y + x] \subseteq \text{Ref}([(2 - x)y + x, (2 - x)y + x]) = \text{Ref}(xy) \) by Theorem 5.4.
(ii) \( \text{cons}_a(x) = ax \subseteq \text{Ref}(ax) = \text{cocons}_a(\text{Ref}(x)) \) from the previous point and since \( \text{cons}_a \) is a particular case of concatenation.
(iii) \( \text{tail}_a(x) = (x/a) \subseteq \text{Ref}(x/a) = \text{tail}_a(\text{Ref}(x)) \) by Theorem 5.4.
(iv) \( \text{head}_a(\text{Ref}(x)) = \text{Ref}(x) <_{F} \text{Ref}(r) \) and by Proposition 5.9, we have \( x <_{F} r \) which equals \( \text{head}_a(x) \). So, \( \text{head}_a(x) \Rightarrow \text{head}_a(x) \).
(v) From the definition of \( \tilde{p}f \)
\[
pf(p, \text{Ref}(x) \text{ else } \text{Ref}(y) = \begin{cases} \text{Ref}(x) & \text{if } p = \text{tt} \\ \text{Ref}(x) \cap_F \text{Ref}(y) & \text{if } p = \perp \\ \text{Ref}(y) & \text{if } p = \text{ff} \end{cases}
\]

and the definition of \( pf \),
\[
pf(p, x \text{ else } y = \begin{cases} x & \text{if } p = \text{tt} \\ x \cap y & \text{if } p = \perp \\ y & \text{if } p = \text{ff} \end{cases}
\]
Since \( x \cap y \subseteq \text{Ref}(x) \cap_F \text{Ref}(y) \), \( x \subseteq \text{Ref}(x) \) and \( x \subseteq x \cap y \) and \( \text{Ref}(x) \subseteq \text{Ref}(x) \cap \text{Ref}(y) \), the result holds. \( \Box \)

The next proposition shows that the obtained floating-point number intervals are less precise than the original ones.

Proposition 6.2.

(i) \( \exists x, y, \text{Ref}(xy) \not\subseteq xy \)
(ii) \( \exists x, \text{cocons}_a(\text{Ref}(x)) \not\subseteq \text{cons}_a(x) \)
(iii) \( \exists x, \text{tail}_a(\text{Ref}(x)) \not\subseteq \text{tail}_a(x) \)
(iv) \( \exists x, \text{head}_a(x) \not= \text{head}_a(\text{Ref}(x)) \)
(v) \( \exists x, y, \text{pf}(p, \text{Ref}(x), \text{Ref}(y)) \not\subseteq \text{pf}(p, x, y) \)
Proof. The proof follows from Theorem 5.5.

(i) \( \exists x, y, \text{Ref} (xy) \nsubseteq xy \) by Theorem 5.5.
(ii) \( \exists x, \text{cons}_a (\text{Ref} (x)) = \text{Ref} (ax) \nsubseteq ax = \text{cons}_a (x) \) by the previous point and since \( \text{cons}_a \) is a particular case of concatenation.
(iii) \( \exists x, \text{tail}_a (\text{Ref} (x)) = \text{Ref} (x) \nsubseteq x/a = \text{tail}_a (x) \) by Theorem 5.5.
(iv) \( \text{head}_a (x) = x < r \neq \text{Ref} (x) \nsubseteq \text{Ref} (r) = \text{head}_a (\text{Ref} (x)) \) because \( x \in \text{Ref} (x) \).
(v) From the definition of \( \text{pif} \) and by Theorem 5.5, the result is straightforward. \( \square \)

Corollary 6.3.

(i) \( xy \subseteq \text{Rep} (\text{Ref} (xy)) \)
(ii) \( \text{cons}_a (x) \subseteq \text{Rep} (\text{cons}_a (\text{Ref} (x))) \)
(iii) \( \text{tail}_a (x) \subseteq \text{Rep} (\text{tail}_a (\text{Ref} (x))) \)
(iv) \( \text{pif} (p, x, y) \subseteq \text{Rep} (\text{pif} (p, \text{Ref} (x), \text{Ref} (y))) \)

Proof. The proof is immediate from Proposition 6.1 since \( x \subseteq \text{Rep} (\text{Ref} (x)) \). \( \square \)

Proposition 6.4.

(i) \( \text{cons}_a (\text{Ref} (x)) \subseteq x_F \Rightarrow \text{cons}_a (x) \subseteq \text{Rep} (x_F) \)
(ii) \( \text{tail}_a (\text{Ref} (x)) \subseteq x_F \Rightarrow \text{tail}_a (x) \subseteq \text{Rep} (x_F) \)
(iii) \( \text{pif} (p, \text{Ref} (x), \text{Ref} (y)) \subseteq z_F \Rightarrow \text{pif} (p, x, y) \subseteq \text{Rep} (z_F) \)

Proof. The proof is immediate from Proposition 6.1. Let us show it for (i). Consider that \( \text{cons}_a (\text{Ref} (x)) \subseteq x_F \) then we have \( \text{Rep} (\text{cons}_a (\text{Ref} (x))) \subseteq \text{Rep} (x_F) \) by monotonicity of the function \( \text{Rep} \), and finally, \( \text{cons}_a (x) \subseteq \text{Rep} (x_F) \) by Corollary 6.3. \( \square \)

The previous proposition ensures the soundness of the basic operators with respect to the \((\text{IL}, \text{IL}_b, \text{Ref}, \text{Rep})\) Galois connection.

Proposition 6.5.

(i) \( \text{cons}_a (x) \subseteq \text{Rep} (x_F) \Rightarrow \text{cons}_a (\text{Ref} (x)) \subseteq x_F \)
(ii) \( \text{tail}_a (x) \subseteq \text{Rep} (x_F) \Rightarrow \text{tail}_a (\text{Ref} (x)) \subseteq x_F \)
(iii) \( \text{pif} (p, x, y) \subseteq \text{Rep} (z_F) \Rightarrow \text{pif} (p, \text{Ref} (x), \text{Ref} (y)) \subseteq z_F \)

Proof. The proof is obtained owing to the Galois connection property \( \text{Ref} \circ \text{Rep} = \text{Id} \). By applying \( \text{Ref} \) on both sides of the equality and since it is monotonic, we get the result. Here is the proof for (i).

(i) For the \( \text{cons}_a \) operator, the proof steps are given by

\[
\text{cons}_a (x) \subseteq \text{Rep} (x_F) \\
\Rightarrow \text{Ref} (\text{cons}_a (x)) \subseteq \text{Ref} (\text{Rep} (x_F)) \text{ by monotonicity of } \text{Ref} \\
\Rightarrow \text{cons}_a (\text{Ref} (x)) \subseteq \text{Ref} (\text{Rep} (x_F)) \text{ since } \text{Ref} (\text{cons}_a (x)) = \text{cons}_a (\text{Ref} (x))
\]

in Definition 6.2 and by Propositions 5.6 and 6.1. \( \square \)
The previous proposition ensures the completeness of the basic operators with respect to the \((I^L, I^L_F, \text{Ref}, \text{Rep})\) Galois connection.

Note that Propositions 6.4 and 6.5 are essential to prove the adequacy property for the basic operations with respect to the Galois connection. That function composition preserves this property as well remains to be proved.

6.3. Composition

For each function \(f\) and \(g\) on \(I^L\), we associate the functions \(\tilde{f}\) and \(\tilde{g}\) on \(I^L_F\) such that \(\text{Ref}(f \circ g(x)) = \tilde{f} \circ \tilde{g}(\text{Ref}(x))\). Then, the function composition translates to the \(I^L_F\) as well.

**Proposition 6.6.** If \(f : I^L \rightarrow I^L\) and \(g : I^L \rightarrow I^L\) are two functions such that \(\forall x \in I^L, f(x) \subseteq \text{Ref}(f(x))\) and \(g(x) \subseteq \text{Ref}(g(x))\), then \(f \circ g(x) \subseteq \text{Ref}(f \circ g(x))\).

**Proof.** Since \(\forall x \in I^L, f(x) \subseteq \text{Ref}(f(x))\) and \(g(x) \subseteq \text{Ref}(g(x))\), then \(f \circ g(x) = f(g(x)) \subseteq \text{Ref}(f(g(x))) = \tilde{f} \circ \tilde{g}(\text{Ref}(x))\) since composition preserves monotonicity. \[\]

**Proposition 6.7.** If \(f, g : I^L \rightarrow I^L\) and \(\tilde{f}, \tilde{g} : I^L_F \rightarrow I^L_F\) such that \(\forall x \in I^L, f(x) \subseteq \tilde{f}(\text{Ref}(x))\) and \(g(x) \subseteq \tilde{g}(\text{Ref}(x))\), then \(f \circ g(x) \subseteq \tilde{f} \circ \tilde{g}(\text{Ref}(x))\).

**Proof.** By Proposition 6.1 and by the monotonicity of the composition. \[\]

**Corollary 6.8.**

\[
\begin{align*}
    f(x) & \subseteq \text{Rep}(\text{Ref}(f(x))) \\
    f \circ g(x) & \subseteq \text{Rep}(\text{Ref}(f \circ g(x))) \\
    f \circ g(x) & \subseteq \text{Rep}(\tilde{f} \circ \tilde{g}(\text{Ref}(x)))
\end{align*}
\]

**Proof.** The proof is immediate from Propositions 5.6, 6.6 and 6.7 and since \(x \subseteq \text{Rep}(\text{Ref}(x))\). \[\]

**Proposition 6.9.**

\[
\tilde{f} \circ \tilde{g}(\text{Ref}(x)) \subseteq y_F \Rightarrow f \circ g(x) \subseteq \text{Rep}(y_F)
\]

**Proof.** Consider that \(\tilde{f} \circ \tilde{g}(\text{Ref}(x)) \subseteq y_F\). Since \(f \circ g(x) \subseteq \tilde{f} \circ \tilde{g}(\text{Ref}(x))\) by Proposition 6.7, we get \(f \circ g(x) \subseteq y_F\) and by definition of \(\text{Rep}\), \(\text{Rep}(y_F) = y_F\), we write \(f \circ g(x) \subseteq \text{Rep}(y_F)\). \[\]

**Proposition 6.10.**

\[
f \circ g(x) \subseteq \text{Rep}(y_F) \Rightarrow \tilde{f} \circ \tilde{g}(\text{Ref}(x)) \subseteq y_F
\]
Proof. Consider that \( f \circ g(x) \subseteq \text{Rep}(y_F) \), then

\[
\Rightarrow \text{Ref}(f \circ g(x)) \subseteq \text{Ref}(\text{Rep}(y_F))
\]

\[
\Rightarrow f \circ g(\text{Ref}(x)) \subseteq y_F \text{ since } \text{Ref}(f \circ g(x)) = f \circ g(\text{Ref}(x))
\]

and by the property of the Galois connection \( \text{Ref}(\text{Rep}(y_F)) = y_F \) of Proposition 5.6. □

The previous two propositions show that function composition preserves the Galois connection property.

6.4. Arithmetic operations on \( I_F \)

Let us consider \( x = [x, \bar{x}] \) and \( y = [y, \bar{y}] \) be two partial real numbers of the domain \( I_F \). The arithmetic operations can be defined on this domain.

- The addition of \( x \) and \( y \) is defined by
  \[
  x +_{I_F} y = [x \oplus y, x \ominus y]
  \]

- The multiplication of \( x \) and \( y \) is defined by
  \[
  x \times_{I_F} y = [x \otimes y, x \oslash y]
  \]

- The inverse of \( x \neq 0 \) is defined by
  \[
  1/_{I_F} x = [1 \odot x, 0 \odot x]
  \]

Roundings: The IEEE 754 standard [10] on floating-point arithmetic defines multiple ways of roundings like rounding to the lowest, to the highest and to the nearest floating-point number.

For the purpose of this work, the operations \(+_{I_F}, \times_{I_F} \) and \(/_{I_F} \) are rounded, if necessary, using the following rules:

- \( x \oplus y, x \otimes y \) and \( 1 \odot x \) are rounded to the lowest floating-point number,
- \( x \ominus y, x \oslash y \) and \( 1 \odot x \) are rounded to the highest floating-point number.

Generally, for an operation on floating-point numbers producing \([z, \bar{z}] \), \( z \) and \( \bar{z} \) are respectively rounded to the lowest and highest floating-point numbers.

Proposition 6.11. The correctness of the arithmetic operations \( \forall x, y \in I^L \) is ensured by

\[
\begin{align*}
 x + y & \subseteq \text{Ref}(x) +_{I_F} \text{Ref}(y) \\
 x \times y & \subseteq \text{Ref}(x) \times_{I_F} \text{Ref}(y) \\
 1/x & \subseteq 1/_{I_F} \text{Ref}(x)
\end{align*}
\]

Note that the operations \(+, \times \) and \(/ \) defined for Real PCF with rationals as end-points can be written in terms of a combination of the basic operations of the language
(\text{cons}_a, \text{tail}_a, \text{head}_a, \text{and } \text{pif}). \text{ See } [7]. \text{ We have given an interpretation with floating-point numbers as end-points that accelerates the computation of these operations as stated above.}

\textbf{Proof.} \text{ By Theorem 5.4 we write that } x \subseteq \text{Ref}(x) \text{ and } y \subseteq \text{Ref}(y), \text{ so we have that } x + y \subseteq \text{Ref}(x) + \text{Ref}(y). \text{ From the definition of the addition of floating-point numbers and the rounding mode we have chosen, we have } \text{Ref}(x) + \text{Ref}(y) \subseteq \text{Ref}(x) + \text{Ref}(y) \text{ and then } x + y \subseteq \text{Ref}(x) + t_s \text{Ref}(y). \text{ The same reasoning applies for the multiplication and for the division.}\]

\textbf{Corollary 6.12.}

\[\forall x, y \in \mathbb{I} \]

\[x + y \subseteq \text{Rep(Ref}(x) +_{t_s} \text{Ref}(y))\]

\[x \times y \subseteq \text{Rep(Ref}(x) \times_{t_s} \text{Ref}(y))\]

\[1/x \subseteq \text{Rep(1/}_{t_s} \text{Ref}(x))\]

\textbf{Proof.} \text{ The proof is immediate from Propositions 5.6, 6.11 and since } x \subseteq \text{Rep(Ref}(x)).\]

The results obtained in this section showed that we can safely replace the operations on partial real numbers with rational end-points by operations on floating-point numbers for real numbers with floating-point numbers as end-points. The accuracy of the representation is decreased, but owing to the hardware representation of the arithmetic operators, the running time is decreased, since these hardware operations take less time.

7. The language of real numbers with floating-point numbers as end-points

The previous sections have presented all the propositions and theorems allowing to safely map partial real numbers with rationals as end-points to partial real numbers with floating-point numbers as end-points and all the related operations. At this level, it is possible to describe the semantics of the Real PCF language which uses this representation of partial real numbers on $\mathbb{I}^k$ instead of the one on $\mathbb{I}^u$.

This section introduces the language PCF where real numbers are represented by intervals with floating-point numbers as end-points. We will give the obtained denotational semantics of the language and show that the implementation is sound with respect to the representation with intervals with rational numbers as end-points.
7.1. Semantics of the of real PCF with floating-point numbers as end-points

7.1.1. Domains

The domain of interpretation $\bigcup \{D_\sigma\}$ is extended to contain the domain $I^\ell_F$ previously introduced.

7.1.2. Interpretation and environments

Environments $\rho_F : \text{var} \rightarrow \bigcup \{D_\sigma\}$ link the variables to their corresponding values. The partial real number variables are linked to values in $I_F$.

The function $A_F : L \rightarrow \bigcup D_\sigma$ of a language $L$ defines the interpretation of the real number constructs in $I^\ell_F$ by

$$A_F[\text{cons}_a] = \text{cohs}_a$$
$$A_F[\text{tail}_a] = \text{tail}_a$$
$$A_F[\text{head}_\ell] = \text{head}_\ell$$
$$A_F[\text{pif}_{p, x, y}] = p_f(p, x, y)$$

The interpretation function maps all the basic operators to the operators defined on $I_F$.

7.1.3. The denotational semantics

The denotational semantics is given by the meaning function $\hat{A}_F : \text{Terms} \rightarrow \text{env} \rightarrow \bigcup D_\sigma$ and is defined by

$$\hat{A}_F[x]_{\rho_F} = \rho_F(x)$$
$$\hat{A}_F[c]_{\rho_F} = A_F[c]$$
$$\hat{A}_F[M \cdot N]_{\rho_F} = \hat{A}_F[M]_{\rho_F}(\hat{A}_F[M]_{\rho_F})$$
$$\hat{A}_F[\lambda x M]_{\rho_F,x} = \hat{A}_F[M]_{\rho_F[x/\epsilon]} \quad \text{with } x \in D_\sigma \text{ if } \epsilon : \sigma$$

The denotational semantics of Real PCF does not change, except that the interpretation of the basic real number operations have been changed to the ones on floating-point numbers as end-points.

7.2. Approximation

Before proving the soundness of the implementation, in PCF extended with real numbers as intervals with floating point numbers as end-points, let us prove that the interpretation $A_F$ in the domain $I^\ell_F$ is sound and gives correct approximations with respect to the interpretation $A$ in the domain $I^L$.

**Proposition 7.1.** For any basic construction $c$ corresponding to $\text{cons}_a$, $\text{tail}_a$, $\text{head}_\ell$, and $\text{pif}$ we have

$$A[c] \subseteq A_F[c]$$
Proof. From the definitions of $A$ and $A_F$, we write that $A[c] = c$ and $A_F[c] = \bar{c}$, and by Proposition 6.1, this result is straightforward. □

**Theorem 7.2.** Real PCF with floating-point numbers as end-points satisfies the following property:

$$\hat{A}[M] \subseteq \hat{A}_F[M]$$

Proof. The proof that $\hat{A}[M] \subseteq \hat{A}_F[M]$ is achieved by structural induction.
- If $\alpha$ is a variable referring to a real number, then $\alpha \subseteq \text{Ref}(\alpha)$ and therefore $\rho(\alpha) \subseteq \rho_F(\text{Ref}(\alpha))$ and then $\hat{A}[\alpha]_\rho \subseteq \hat{A}_F[\alpha]_\rho$;
- $\hat{A}[c]_\rho \subseteq \hat{A}_F[c]_\rho$, by Proposition 7.1;
- let us assume that $\hat{A}[M]_\rho \subseteq \hat{A}_F[M]_\rho$ and $\hat{A}[N]_\rho \subseteq \hat{A}_F[N]_\rho$. Since composition is a monotonic operation and by Proposition 6.7, we can write $\hat{A}[M]_\rho(\hat{A}[N]_\rho) \subseteq \hat{A}_F[M]_\rho(\hat{A}_F[N]_\rho)$, and then we write that $\hat{A}[MN]_\rho \subseteq \hat{A}_F[MN]_\rho$;
- let us assume that $\hat{A}[M]_\rho \subseteq \hat{A}_F[M]_\rho$. If $\alpha$ is a free real variable in $M$ and since $\hat{A}[\alpha]_\rho \subseteq \hat{A}_F[\alpha]_\rho$, and since abstraction preserves monotonicity, we can assert that $\hat{A}[\lambda x. M]_\rho \subseteq \hat{A}_F[\lambda x. M]_\rho$.

The assertion has been proved to be valid on each PCF program construction, then it is valid for any program built by the composition of these constructions. □

### 7.3. Soundness

The soundness of the refinement of $I^L$ by $I_F^L$ is defined by

$$\forall x_F \in I_F^L, \hat{A}_F[M]_{\bot_L} \subseteq x_F \Rightarrow \hat{A}[M]_{\bot} \subseteq \text{Rep}(x_F)$$

for any program $M$ denoting a real program.

**Proposition 7.3.** For any real program basic construction $c$ corresponding to cons\(_a\), tail\(_a\), and pif,

$$\forall x_F \in I_F^L, \ A_F[c] \subseteq x_F \Rightarrow A_F[c] \subseteq \text{Rep}(x_F)$$

Proof. The proof is directly provided by Proposition 6.4 and since $A[c] = c$ and $A_F[c] = \bar{c}$. □

**Theorem 7.4 (Soundness).** For any term $M$ denoting a real program,

$$\forall x_F \in I_F^L, \hat{A}_F[M]_{\bot_L} \subseteq x_F \Rightarrow \hat{A}[M]_{\bot} \subseteq \text{Rep}(x_F)$$
Proof. The proof is achieved by a structural induction on the terms of the language.

(i) If $x$ is a variable referring to a real number, then,

$$\hat{A}_F[x] \subseteq x_F \Rightarrow \rho_F(x) \subseteq x_F$$

$$\Rightarrow \text{Rep}(\rho_F(x)) \subseteq \text{Rep}(x_F)$$

$$\Rightarrow [\rho_F(x), \rho_F(x)] \subseteq \text{Rep}(x_F)$$

$$\Rightarrow \rho(x) \subseteq \text{Rep}(x_F) \quad \text{since} \ \rho(x) \subseteq \rho_F(x)$$

(ii) Immediate by Proposition 7.3.

(iii) Let us assume that for $y_F \in I^L, \hat{A}_F[M]_{y_F} \subseteq x_F \Rightarrow \hat{A}[M]_{\rho_{\text{Rep}(y_F)}} \subseteq \text{Rep}(x_F)$ and $\hat{A}_F[N]_{y_F} \subseteq y_F \Rightarrow \hat{A}[N]_{\rho} \subseteq \text{Rep}(y_F)$, then by Proposition 6.7 and since composition preserves monotonicity, we have $\hat{A}_F[MN]_{y_F} \subseteq x_F \Rightarrow \hat{A}[MN]_{\rho} \subseteq \text{Rep}(x_F)$.

(iv) Assume that $\hat{A}_F[M]_{y_F} \subseteq x_F \Rightarrow \hat{A}[M]_{\rho} \subseteq \text{Rep}(x_F)$ then, if $\alpha$ is a free real variable such that $\hat{A}_F[M]_{y_F} \subseteq y_F \Rightarrow \hat{A}M_{\rho} \subseteq \text{Rep}(y_F)$, and since abstraction preserves monotonicity, $\hat{A}_F[\lambda x.M]_{y_F} \subseteq x_F \Rightarrow \hat{A}[\lambda x.M]_{\rho_{\text{Rep}(y_F)/x}} \subseteq \text{Rep}(x_F)$.

7.4. Completeness

The completeness of the representation of $I^L$ by $I^F_F$ is defined by

$$\forall x_F \in I^L, \hat{A}[M]_{\perp} \subseteq \text{Rep}(x_F) \Rightarrow \hat{A}_F[M]_{\perp} \subseteq x_F$$

for any term $M$ denoting a real program.

Proposition 7.5. For any real program basic construction $c$ corresponding to $\text{cons}_u$, tail, and $\text{pif}$,$$
\hat{A}[c] \subseteq \text{Rep}(x_F) \Rightarrow A_F[c] \subseteq x_F$$

Proof. The proof results from Proposition 6.6 and since $\hat{A}[c] = c$ and $A_F[c] = \bar{c}$. 

Theorem 7.6 (Completeness). For any term $M$ denoting a real program,

$$\forall x_F \in I^L, \hat{A}[M]_{\perp} \subseteq \text{Rep}(x_F) \Rightarrow \hat{A}_F[M]_{\perp} \subseteq x_F$$

Proof. The proof is achieved by a structural induction on the terms of the language.

(i) If $x$ is a variable, then,

$$\hat{A}[x]_{\rho} \subseteq \text{Rep}(x_F) \Rightarrow \rho(x) \subseteq \text{Rep}(x_F)$$

$$\Rightarrow \text{Rep}(\rho(x)) \subseteq \text{Rep}(\text{Rep}(x_F))$$

$$\Rightarrow \hat{A}_F[x]_{\rho} \subseteq x_F \quad \text{since} \ \text{Rep}(\text{Rep}(x_F)) \subseteq x_F$$

(ii) Immediate by Proposition 7.5.

(iii) Let us assume $y_F \in I^L$ and $\hat{A}[M]_{x_F} \subseteq \text{Rep}(x_F) \Rightarrow \hat{A}_F[M]_{y_F} \subseteq x_F$ and $\hat{A}[N]_{y_F} \subseteq \text{Rep}(y_F) \Rightarrow \hat{A}_F[N]_{y_F} \subseteq y_F$ then by Proposition 6.8 and by monotonicity of the application, we obtain $\hat{A}[MN]_{y_F} \subseteq \text{Rep}(x_F) \Rightarrow A_F[MN]_{y_F} \subseteq x_F.$
(iv) Assume that $\hat{A}[M]_p \subseteq \text{Rep}(x_F)$, then if $x$ is a free variable such that there exists $y_F \in I_F$ with $\hat{A}[z]_p \subseteq \text{Rep}(y_F) \Rightarrow \hat{A}[z]_p \subseteq y_F$ then, since abstraction preserves monotonicity, we get $\hat{A}[\lambda z.M]_p[\text{Rep}(y_F); x] \subseteq \text{Rep}(x_F) \Rightarrow \hat{A}[\lambda z.M]_p[\text{Rep}(y_F); x] \subseteq x_F$. □

7.5. Adequacy property

Theorems 7.4 and 7.6 prove respectively that the soundness and completeness properties are guaranteed by the suggested representation of partial real numbers with rationals as end-points by partial real numbers with floating-point numbers as end-points. Moreover, these properties apply to the PCF extension suggested in [7]. Therefore, the adequacy property with respect to the defined abstract interpretation is ensured.

7.6. Evaluation, accuracy of the representation and program analysis

Program analysis techniques seem to be well adapted when trying to solve the problem of accuracy of the representation by floating-point numbers. Practically, the number of digits is fixed by a precision expressed by the user. By precision, we mean the length $|x - \bar{x}|$ of the interval $[\bar{x}, \bar{x}]$. This precision fixes the number of digits needed on the end-points in order to give them a finite representation. This finite representation will ensure the termination of the computation of the end-points, but not the termination of a program (due to the $\text{pref}$ instruction).

In the context of floating-point numbers representation, the minimum precision is known and is constant. It is equal to the difference between two consecutive floating-point numbers and is noted $e_F$.

The program analysis has to answer to the following question: when is it possible to represent real numbers with rational end-points with a given precision $\varepsilon = |x - \bar{x}|$ by floating-point numbers as end-points with a precision $e_F \leq \varepsilon$? We do not give the whole details of this program analysis. It is developed in another paper [1] but, we give a survey of this analysis in order to show to the reader the feasibility of this analysis.

Let us assume $x = [\bar{x}, \bar{x}]$ to be a real number with rational numbers as end-points. Let $l_x$ be the number of the same digits representing $\bar{x}$ and $\bar{x}$. Then, the precision $\varepsilon$ associated to the partial real number $x = [\bar{x}, \bar{x}]$ in $I^L$ satisfies $B^{-(l_x + 1)} \leq \varepsilon \leq B^{-l_x}$, where $B$ is the base where the number $x$ is represented and $l_x$ is the length of the sequence of the same digits of $\bar{x}$ and $\bar{x}$.

On the other hand, let $x_F = \text{Ref}([x, \bar{x}]) = [x_F, \bar{x}_F]$ in $I_F$ be the representation of $x$ by floating-point numbers as end-points, and let $(l_m, v_e)$ and $(\bar{l}_m, \bar{v}_e)$ be the length of the mantissa and the value of the exponent associated to $x_F$ and $\bar{x}_F$ respectively. We note $\text{len}_m$ the number of the same digits in the mantissas of $x_F$ and $\bar{x}_F$. If we extract from $x_F$ and $\bar{x}_F$ the corresponding rational numbers, then the number $l_{x_F}$ of the same digits is given by: $l_{x_F} = \text{If } v_e = \bar{v}_e \text{ then } |v_e| + \text{len}_m, \text{ else } \text{min}(v_e, \bar{v}_e)$.

And finally, we get $B^{-(l_{x_F} + 1)} \leq e_F \leq B^{-l_{x_F}}$. 

Definition 7.7. If $e_F \leq \varepsilon$ then the representation of $x$ is correct with respect to the precision given to $x$.

Proposition 7.8. Let $x_F = \text{Ref}(x) = \text{Ref}([x,\bar{x}]) = [x_F, \bar{x}_F]$ be the representation of $x$ by floating-point numbers as end-points. If $l_x \leq l_{x_F}$ then the representation of $x$ is correct with respect to the precision given to $x$.

Proof. By the previous relations, we have $B^{-l_x+1} \leq \varepsilon \leq B^{-l_x}$ and $B^{-l_{x_F}+1} \leq e_F \leq B^{-l_{x_F}}$. If $l_x \leq l_{x_F}$, it implies that $[B^{-l_x+1}, B^{-l_x}] \subseteq [B^{-l_{x_F}+1}, B^{-l_{x_F}}]$. □

The previous proposition gives a sufficient condition to implement partial real numbers with rationals as end-points by real numbers with floating-point numbers as end-points. It allows to guide the choice of real number representations by taking into account the suited accuracy [1]. Moreover, if this accuracy is not reached, then other implementations based on lists [3, 13, 18] can be suggested. This is an important topic allowing to control the formal development of safe numerical software and mainly the choice of data representation.

8. Conclusion

Real number computation is an important topic in the area of numerical computation and safety of critical systems. The representation of numbers in computers commonly used is by means of floating-point numbers. These numbers neither give precise results (due to cancellations and overflows) nor a complete representation of real numbers.

This paper has shown a complete and practical development for the real numbers data type. It is made of three main development steps. The first step recalls the basic mathematical specification of real numbers (B-adic numbers, Cauchy sequences, ...). The second step presented a first refinement by intervals with rationals as end-points represented by lists of digits with cons, head, tail · · · operators. At last, the third step refines the previous implementation by intervals with floating-point numbers as end-points which uses the hardware implementation of floating-point numbers and therefore increases the running time performance. The proof of correctness of the transformations performed during this development have been given as well.

The origin of this paper is real number computation, and mainly the extension of PCF to handle real numbers as intervals with rationals as end-points. This paper has presented a combination of the use of floating-point numbers arithmetic with real number computation. It has shown an implementation of real numbers as intervals with rational numbers as end-points by real numbers with floating-point numbers as end-points. This representation is useful from the point of view of implementation and of efficiency although a weak version of the adequacy property is proved. Indeed, the representation we have presented on the kernel of the PCF language, has been proved to be sound and complete with respect to the defined abstract interpretation.
The soundness property ensures that for any computation of a partial real number, with floating-point numbers as end-points, we always get a partial real number with floating-point numbers as end-points which is a correct approximation of the exact real number. This property states the correctness of our suggested implementation.

On the other hand, the finite number of floating-point numbers allowed on a given machine makes the accuracy of this representation to be constant as opposed to the rational numbers representation whose accuracy can be parametrised. This means that there does not exist a one-to-one mapping between real numbers with rational end-points and real numbers with floating-point numbers as end-points. In other words, the representation of real numbers with rational end-points is more accurate since the end-points can have an arbitrary and an infinite number of digits.

In order to solve the problem of precision of the representation, we recalled the basis of the program analysis technique developed in [1]. It allows to choose between possible implementations of partial real numbers either by intervals with floating-point numbers as end-points or with rationals with a finite number of digits as end-points. This analysis allows to introduce control during formal development of numerical software. A comparison of the suited precision to one of the floating-point numbers implementation allows us to choose the correct representation and avoid the use of list implementations when the required precision is bigger than the maximal precision allowed by the use of floating-point numbers.

Finally, we plan to investigate the issue of supporting the proving process of the refinement achieved in this paper by a prover. We also plan to code all the proofs in the higher order type theorem prover DEVA [20, 21].

References