Bachelor Thesis

Program extraction for exact real numbers: Stream multiplication.

Till Überrück-Fries
till@ueberfries.de
Matriculation number: 10968924

submitted to the

Department of Mathematics
Faculty of Mathematics, Informatics and Statistics
Ludwig-Maximilians-Universität München

July 2016

supervised by
Prof. Dr. Helmut Schwichtenberg

Bachelorarbeit

Programmextraktion für exakte reelle Zahlen: Multiplikation von Strömen.

Till Überrück-Fries
till@ueberfries.de
Matrikelnummer: 10968924

vorgelegt dem

Mathematischen Institut
Fakultät für Mathematik, Informatik und Statistik
Ludwig-Maximilians-Universität München

Juli 2016

bei

Prof. Dr. Helmut Schwichtenberg

Abstract

Signed digit code is a representation of the real numbers by infinite streams. As infinite data types they allow computation to any desired degree of exactitude. To obtain reliable algorithms, I use the method of program extraction. I present a formal proof about real multiplication in an appropriate formal theory involving inductive and coinductive definitions. I extract real number algorithms for the multiplication function of signed digit streams. The proof is formalised in the proof assistant Minlog using a concrete implementation of the reals by cauchy sequences with modulus.

Signed Digit Code ist eine Darstellung der reellen Zahlen durch unendliche Ströme. Als unendliche Datentypen erlauben sie Rechnungen zu beliebiger Genauigkeit durchzuführen. Um zuverlässige Algorithmen zu erhalten, wende ich die Methode der Programmextraktion an. Ich liefere einen formalen Beweis über die reelle Multiplikation in einer geeigneten formalen Theorie, die induktive und koinduktive Definitionen erlaubt. Ich extrahiere reellwertige Algorithmen für die Multiplikation von Signed Digit Strömen. Der Beweis ist mit dem Beweisassistenten Minlog formalisiert für eine konkrete Implementierung der reellen Zahlen als Cauchy-Folgen mit Modulus.

Contents

1 Introduction **5**

2 Signed digit streams **6**

2.1 An algebra of signed digit streams 6

2.2 The predicate ${}^{co}I$ 7

2.3 Realisability 8

3 The informal proof **9**

4 Formalisation **13**

4.1 ...for concrete reals 14

4.2 ...for abstract reals 20

5 Extraction **21**

5.1 Corecursion 21

5.2 Minlog notation 21

5.3 The extracted terms 22

6 Conclusion **28**

References **28**

1 Introduction

In the light of an increasing importance of high-precision computer programs, a reliable implementation of the real numbers is desirable. Exact real number computation requires: (a) a dependable implementation of real numbers, and (b) the formal verification of the computational algorithms. To address (b), real numbers need to be implemented in a logical framework, that provides the means to carry out rigorous proofs. The usual approach is to first give a concrete implementation and prove the correctness in a second step. An example of this approach can be found in (Ciaffaglione and Gianantonio, 2006).

A different approach has been proposed in (Berger and Seisenberger, 2010): to provide an abstract implementation of the real numbers, where the realisability is guaranteed by the underlying formal system, the "theory of computable functionals" (TCF, see (Schwichtenberg and Wainer, 2012)). Then, a proof using minimal logic on the abstract real numbers implicitly provides algorithms for the computation on concrete real numbers.

In this thesis, I aim to give an example of the practicality of this approach. Extending the works about the average function by Miyamoto and Schwichtenberg (2015) and by Berger et al. (2016), I will extract algorithms for stream multiplication of signed digit streams.

More explicitly, I will present a proof of

$$\forall^{\mathrm{nc}}_{x,y\in {}^{co}I}(-\frac{1}{4} < \frac{x\cdot y}{4} < \frac{1}{4} \to {}^{co}I(x\cdot y)), \tag{1}$$

where ${}^{co}I$ is a coinductively defined predicate meaning: if x is in ${}^{co}I$, then there exists an infinite stream of signed digits representing x. The decorated quantifier denotes that x and y are not computationally relevant. It is a consequence of the approach to move the computational content to the predicate ${}^{co}I$.

An informal proof can be found in section 3. This proof is carried out in the interactive proof assistant Minlog[1] for concrete real numbers. To show the difference of the approach, this is also partially done for abstract real numbers. Both versions are discussed in section 4 with an emphasise on concrete reals. In section 5, I present the extracted terms of the concrete version.

[1] See `www.minlog-system.de/`.

2 Signed digit streams

Signed digit code is a representation of the unit interval $\mathbb{I} = [-1, 1]$ as infinite streams of $\mathbf{SD} = \{-1, 0, 1\}$ (signed digits) so that $d_1 d_2 d_3 \ldots$ represents

$$\sum_{i=1}^{\infty} \frac{d_i}{2^i}. \tag{2}$$

2.1 An algebra of signed digit streams

We consider the free algebra $\mathbf{I}$ of "standard intervals", given by a nullary constructor $\mathsf{nil}_{\mathbf{I}}$ and a binary constructor C of type $\mathbf{SD} \to \mathbf{I} \to \mathbf{I}$; we write $C_d v$ for $C d v$.

$$\mathbf{I} = \mathsf{nil}_{\mathbf{I}} + C\ \mathbf{SD}\ \mathbf{I}$$

A finite constructor expression $v = C_{d_0}(C_{d_1} \ldots (C_{d_{k-1}} \mathbb{I}) \ldots)$ can be seen as a "total ideal" of the algebra $\mathbf{I}$. The intuition is that v denotes the subinterval of $\mathbb{I}$ that includes all reals with a signed digit representation that start with $d_0 d_1 \ldots d_{k-1}$.

For v as above, we define the interpretation function $[\![\ ]\!] : \mathbf{I} \to \mathbb{I}$ by recursion:

$$[\![\mathbb{I}]\!] = \mathbb{I},$$
$$[\![C_d(v_0)]\!] = \frac{[\![v_0]\!] + d}{2}.$$

For example, $[\![C_{-1}\mathbb{I}]\!] = [-1, 0]$, $[\![C_0\mathbb{I}]\!] = [-\frac{1}{2}, \frac{1}{2}]$ and $[\![C_0(C_1\mathbb{I})]\!] = [0, \frac{1}{2}]$.

For a real number in $\mathbb{I}$, the signed digit stream $d_1 d_2 d_3 \ldots$ corresponds to an infinite constructor expression $v = C_{d_1}(C_{d_2}(C_{d_3}(\ldots))) \in \mathbf{I}$. The real number then lies in the infinite intersection of all intervals denoted by the finite truncations of v. We see an infinite constructor expression such as v as a "cototal" ideal of the algebra $\mathbf{I}$.

It becomes clear that, for the purpose of signed digit code, we are only interested in these cototal ideals. Actually, we can still speak of a cototal ideal when the underlying algebra does *not* have a nullary constructor. For the purpose of simpler proofs, we can drop the nullary constructor. The algebra $\mathbf{I}$ becomes

$$\mathbf{I} = C\ \mathbf{SD}\ \mathbf{I}.$$

The interpretation function is now defined by corecursion:

$$[\![C_d(v_0)]\!] = \frac{[\![v_0]\!] + d}{2}.$$

2.2 The predicate ${}^{co}I$

Our goal is to extract a term that denotes a function taking the stream representations of two real numbers x, y as input and giving out the stream representation of their product $x \cdot y$. The idea is to carry out a constructive proof of a formula A that provides us with such a function as its witness. Therefore, the function should have the type $\mathbf{I} \to \mathbf{I} \to \mathbf{I}$. To achieve this result, we need to be careful about the computational content of the formula A. It must not rely on the type of x and y, i.e. the concrete implementation of real numbers, but only on their signed digit streams. By using non-computational universal quantifiers $\forall^{\mathrm{nc}}$, we move the computational content from the quantified variable to the kernel of the formula. We have to keep this in mind for the definition of ${}^{co}I$.

Now define inductively the set I of (abstract) reals, by the I-clause

$$\forall_x^{\mathrm{nc}} \forall_d (I(x) \to I\left(\frac{x+d}{2}\right)) \tag{3}$$

where $d \in \mathbf{SD}$. The quantifier $\forall^{\mathrm{nc}}$ represents the fact, that we are not interested in x as input, but only in how to obtain a new element of I from the signed digit d. Note that in consequence of our omittance of a nullary constructor for $\mathbf{I}$, the set I has no initial clause and is not well-founded.

For I we define coinductively the predicate ${}^{co}I$ by the ${}^{co}I$-clause

$$\forall_x^{\mathrm{nc}} ({}^{co}I(x) \to \exists_y^{\mathrm{r}} \exists_d (x = \frac{y+d}{2} \wedge {}^{co}I(y))). \tag{4}$$

Analogously to $\forall^{\mathrm{nc}}$, the quantifier $\exists^{\mathrm{r}}$ denotes that y is not considered computationally relevant.

The existence of I and ${}^{co}I$ is guaranteed by our underlying theory TCF. To see this consider the operator

$$\Phi(X) := \{x \mid \exists_{x' \in X}^{\mathrm{r}} \exists_d (x = \frac{x'+d}{2})\},$$

where $d \in \mathbf{SD}$. Since $\Phi(X)$ is strictly positive in X, TCF provides us with the sets I and ${}^{co}I$ for the least and greatest fixed point of Φ:

$$\begin{aligned} I &:= \mu_X \Phi(X) && \text{least fixed point,} \\ {}^{co}I &:= \nu_X \Phi(X) && \text{greatest fixed point} \end{aligned}$$

satisfying the (strengthened) axioms

$$\begin{aligned} \Phi(I \cap X) \subseteq X \to I \subseteq X \qquad & \text{induction,} \\ X \subseteq \Phi({}^{co}I \cup X) \to X \subseteq {}^{co}I \qquad & \text{coinduction.} \end{aligned}$$

We reformulate the greatest-fixed-point axiom to

$$\forall_x^{\mathrm{nc}}(P(x) \to \forall_y^{\mathrm{nc}}(P(y) \to \exists_z^{\mathrm{r}}\exists_d(y = \frac{z+d}{2} \land ({}^{co}I(z) \lor P(z)))) \to {}^{co}I(x)), \tag{5}$$

where P is a predicate denoting a set of real numbers. (5) says that P is a subset of ${}^{co}I$.

An immediate consequence of the greatest-fixed-point axiom is that the inverse ${}^{co}I$-clause holds as well:

$$\forall_x^{\mathrm{nc}}(\exists_y^{\mathrm{r}}\exists_d({}^{co}I(y) \land x = \frac{y+d}{2}) \to {}^{co}I(x)). \tag{6}$$

2.3 Realisability

To extract a program from a proof one essentially removes computationally irrelevant parts from the proof. This method is based on *modified realisability* as described in detail in (Schwichtenberg and Wainer, 2012). We say a program et(M) "realises" a formula A if it provides to every input (depending on the formula) a witness for the validity of A. The specific extraction rules used in this thesis are discussed in (Miyamoto and Schwichtenberg, 2015, p. 1696).

The realiser of our proof about multiplication will transform the streams of x and y to a stream of $x \cdot y$. To better express the fact that a signed digit stream v represents a real number x we introduce the realizability extensions $I^{\mathbf{r}}$ and $({}^{co}I)^{\mathbf{r}}$. They are binary predicates on signed digit streams v (resulting from $\exists_d$ in the definition of $\Phi(x)$) and real numbers x. Consider the operator

$$\Phi^{\mathbf{r}}(Y) := \{(v,x) \mid \exists^{\mathrm{u}}_{(v',x')\in Y}\exists_d(x = \frac{x'+d}{2} \land v = C_d(v'))\}$$

(the quantifier $\exists^{\mathrm{u}}$ means that neither the quantified variable nor the kernel is computationally significant). $\Phi^{\mathbf{r}}(Y)$ is strictly positive in Y, so as for Φ, TCF provides us with predicates $I^{\mathbf{r}}$ and $({}^{co}I)^{\mathbf{r}}$ for the least and greatest fixed point of $\Phi^{\mathbf{r}}$:

$$\begin{aligned} I^{\mathbf{r}} &:= \mu_Y \Phi^{\mathbf{r}}(Y) \qquad && \text{least fixed point,} \\ ({}^{co}I)^{\mathbf{r}} &:= \nu_Y \Phi^{\mathbf{r}}(Y) \qquad && \text{greatest fixed point} \end{aligned}$$

satisfying the (strengthened) axioms

$$\Phi^{\mathbf{r}}(I^{\mathbf{r}} \cap Y) \subseteq Y \to I^{\mathbf{r}} \subseteq Y \qquad \text{induction},$$
$$Y \subseteq \Phi^{\mathbf{r}}((^{co}I)^{\mathbf{r}} \cup Y) \to Y \subseteq (^{co}I)^{\mathbf{r}} \qquad \text{coinduction}.$$

In (Berger et al., 2016, p. 16) the correctness of the definition of ^{co}I is proven in the sense that the witnesses of $^{co}I(x)$ are exactly the signed digit representations of x.

Proposition. *For* $s = d_1 d_2 d_3 \ldots \in \mathbf{SD}^\omega$, $v = C_{d_1}(C_{d_2}(C_{d_3}(\ldots))) \in \mathbf{I}$ *and* $x \in \mathbb{I}$

$$(^{co}I)^{\mathbf{r}}(s, x) \leftrightarrow x = [\![v]\!].$$

This ensures that the program extracted from (1) takes the witnesses s_x of $^{co}I(x)$ and s_y of $^{co}I(y)$ and produces a witness s of $^{co}I(x \cdot y)$, i.e. a signed digit stream representing the product of x and y.

3 The informal proof

I now present an informal proof of (12) about the multiplication function for signed digit streams. The proof follows the work of Miyamoto and Schwichtenberg (2015) about the average function. It relies on an appropriate auxiliary function which was presented by Ciaffaglione and Gianantonio (2006, p. 43). In section 4, this proof will be formalised for concrete and partially for abstract real numbers. By convention, we use the variable names d, e for integers with $|d|, |e| \leq 1$ and i, j for integers with $|i|, |j| \leq 2$.

Theorem (Multiplication)**.**

$$\forall^{\mathrm{nc}}_{x,y \in {}^{co}I}(-\frac{1}{4} < \frac{x \cdot y}{4} < \frac{1}{4} \to {}^{co}I(x \cdot y))$$

Proof. Consider the sets

$$X := \{\frac{x \cdot y}{4} \mid x, y \in {}^{co}I\}, \qquad Y := \{\frac{x \cdot y + z + i}{4} \mid x, y, z \in {}^{co}I, |i| \leq 2\}.$$

We will show that $X \subseteq Y$ and that Y satisfies the clause coinductively defining ^{co}I. By the greatest-fixed-point-axiom for ^{co}I, we get $Y \subset {}^{co}I$ and hence, $X \subseteq {}^{co}I$. We will then show that for $-\frac{1}{2} < x < \frac{1}{2}$ and $^{co}I(x)$, also $^{co}I(2 \cdot x)$. Applying this twice to the elements of X gives our claim. □

Why do we need the additional assumption $-\frac{1}{4} < \frac{x \cdot y}{4} < \frac{1}{4}$? It seems that without

a nullary constructor of **I**, we cannot prove ${}^{co}I \subseteq \mathbb{I}$. However, we need to ensure in the proof of ${}^{co}I(x) \to {}^{co}I(2 \cdot x)$, that we stay within $\mathbb{I}$ by assuming $-\frac{1}{2} < x < \frac{1}{2}$. Unfortunately, $-\frac{1}{2} \leq x \leq \frac{1}{2}$ does not suffice as we will see later. In consequence, the multiplication theorem does not cover the cases where $x, y \in \{-1, 1\}$. However, since $x, y \in \{-1, 1\}$ have unique stream representations, it should be easy to prove that the extracted program is also correct for $x, y \in \{-1, 1\}$.

Next, we show three auxiliary propositions that we will need later.

Sublemma (`CoIZero`)**.**

$$ {}^{co}I(0) \tag{7}$$

Proof. Let $P := \{x \mid x = 0\}$. By the greatest-fixed-point axiom (5), we need to show $\forall_y^{\mathrm{nc}}(P(y) \to \exists_z^{\mathrm{r}} \exists_d (y = \frac{z+d}{2} \land {}^{co}I(z) \lor P(z)))$. Because $y = 0$ is the only element of P, we can choose $z = 0 \in P$ and $d = 0$. Now, $y = 0 = \frac{0+0}{2} = \frac{z+d}{2}$ and $P(z)$ satisfies the right hand side of the disjunction. □

Sublemma (`CoIMinus`)**.**

$$ \forall_x^{\mathrm{nc}}({}^{co}I(-x) \to {}^{co}I(x)) \tag{8}$$

Proof. Let $P := \{x \mid {}^{-}x \in {}^{co}I\}$ and $x \in P$. By the greatest-fixed-point axiom (5), it suffices to show $\exists_z^{\mathrm{r}} \exists_d (x = \frac{z+d}{2} \land ({}^{co}I(z) \lor P(z)))$. Since $x \in P$, there exists $y \in {}^{co}I$ and d such that ${}^{-}x = \frac{y+d}{2}$. Now $x = \frac{{}^{-}y-d}{2}$ and $P({}^{-}y)$ satisfies the right hand side of the disjunction, since ${}^{-}({}^{-}y) = y \in {}^{co}I$. □

Sublemma (`CoITimesD`)**.**

$$ \forall_x^{\mathrm{nc}} \forall_d ({}^{co}I(x) \to {}^{co}I(d \cdot x)) \tag{9}$$

Proof. Let $x \in {}^{co}I$. We distinguish cases on d.

$d = {}^{-}1$: We need to show ${}^{co}I(-x)$. This follows from `CoIMinus`, since ${}^{-}(-x) = x \in {}^{co}I$.

$d = 0$: We need to show ${}^{co}I(0)$. We use `CoIZero`.

$d = 1$: We need to show ${}^{co}I(x)$, which follows from the premise.

□

Furthermore, we need a result about the auxiliary average function $\frac{x+y+i}{4}$, that was shown in (Miyamoto and Schwichtenberg, 2015, p. 1699).

Lemma (AverageAuxSatClause)**.**

$$ \forall_{x,y \in {}^{co}I}^{\mathrm{nc}} \forall_i \; {}^{co}I\left(\frac{x+y+i}{4}\right) \tag{10}$$

Lemma (`CoIMultToMultc`, XSubY)**.**

$$\forall^{\mathrm{nc}}_{x,y\in{}^{co}I}\exists_i\exists^{\mathrm{r}}_{z\in{}^{co}I}\ \frac{x\cdot y}{4}=\frac{x\cdot y+z+i}{4} \tag{11}$$

Proof. Let $x, y \in {}^{co}I$. Naturally, assume $i = 0$ and $z = 0 \in {}^{co}I$ (due to `CoIZero`). Then $\frac{x\cdot y}{4} = \frac{x\cdot y+0+0}{4} = \frac{x\cdot y+z+i}{y}$. □

The following is the central idea of the proof. We show that Y satisfies the premise of the greatest-fixed-point axiom (5): $\forall^{\mathrm{nc}}_{y\in Y}\exists^{\mathrm{r}}_z\exists_d(y = \frac{z+d}{2} \wedge ({}^{co}I(z) \vee P(z)))$. For the formulation of the lemma, I have resolved the definition of Y and chosen the right hand side of the conjunction.

Lemma (`CoIMultcSatCoIClause`)**.**

$$\forall_i\forall^{\mathrm{nc}}_{x,y,z\in{}^{co}I}\exists_{j,d}\exists^{\mathrm{r}}_{x',y',z'\in{}^{co}I}\ \frac{x\cdot y+z+i}{4}=\frac{\frac{x'\cdot y'+z'+j}{4}+d}{2}. \tag{12}$$

Proof. Let $|i| \le 2$ and $x, y, z \in {}^{co}I$. Assume for instance $y = y'$, $x = \frac{x'+d_0}{2}$, $z = \frac{z_1+d_1}{2}$ for some $x', z_1 \in {}^{co}I$ and $|d_0|, |d_1| \le 2$. Then

$$\frac{x\cdot y+z+i}{4}=\frac{(x'+d_0)\cdot y'+z_1+d_1+2i}{8}=\frac{x'\cdot y'+(z_1+d_0\cdot y'+i)+d_1+i}{8}. \tag{13}$$

Let $u := d_0 \cdot y'$ and $v := \frac{z_1+u+i}{4}$. Due to `CoITimesD` $u \in {}^{co}I$ and hence, following from `AverageAuxSatClause`, $v \in {}^{co}I$. Assume for instance $v = \frac{v'+e_0}{2} = \frac{\frac{z'+e}{2}+e_0}{2}$ for some $v', z' \in {}^{co}I$ and $|e|, |e_0| \le 1$. Then

$$\frac{x\cdot y+z+i}{4}=\frac{x'\cdot y'+4v+d_1+i}{8}=\frac{x'\cdot y'+z'+d_1+2e_0+e+i}{8}. \tag{14}$$

Solve $d_1 + 2e_0 + e + i = j + 4d$, which is clearly solvable for $|j| \le 2, |f| \le 1$. Then

$$\frac{x\cdot y+z+i}{4}=\frac{x'\cdot y'+z'+j+4d}{8}=\frac{\frac{x'\cdot y'+z'+j}{4}+d}{2}. \tag{15}$$

□

So far we have:

Lemma (`CoIMultToCoI`)**.**

$$\forall^{\mathrm{nc}}_{x,y\in{}^{co}I}\ {}^{co}I\left(\frac{x\cdot y}{4}\right) \tag{16}$$

For the last step, we need two more auxiliary propositions.

Sublemma (`CoIShift`)**.**

$$\forall^{\mathrm{nc}}_{x\in{}^{co}I}(x < -\frac{1}{2} \to {}^{co}I(x+1)) \tag{17}$$

Proof. Let $x \in {}^{co}I$ such that $x < -\frac{1}{2}$. Thereof, we can assume for instance $x = \frac{x'+d}{2}$ for some $x' \in {}^{co}I$ and $d = -1$. By the inverse ${}^{co}I$-clause, it suffices to show $\exists^{\mathrm{r}}_y \exists_e({}^{co}I(y) \wedge x+1 = \frac{y+e}{2})$. Consider

$$x+1 = \frac{x'+d}{2}+1 = \frac{x'-1}{2}+1 = \frac{x'+1}{2}.$$

Hence, we can choose $y = x'$ and $e = 1$. □

Sublemma (`TimesTwoAux`)**.**

$$\forall^{\mathrm{nc}}_{x\in{}^{co}I}(x < 0 \to {}^{co}I(x+1)) \tag{18}$$

Proof. Assume $x \in {}^{co}I$ with $x < 0$. Since $x < 0$, there exists an $n \in \mathbb{N}_0$ such that $x < -\left(\frac{1}{2}\right)^{n+1}$. I will show the stronger proposition

$$\forall_n \forall^{\mathrm{nc}}_{x\in{}^{co}I}(x < -\left(\frac{1}{2}\right)^{n+1} \to {}^{co}I(x+1)).$$

This will do the job.

We use induction on n. Let $x \in {}^{co}I$ be arbitrary.

$n = 0$: Let $x \in {}^{co}I$ with $x < -\frac{1}{2}$. We can now use `CoIShift`.

$n \mapsto n+1$: Let $x \in {}^{co}I$ with $x < -\left(\frac{1}{2}\right)^{n+2}$. From ${}^{co}I(x)$ we get

$$\exists^{\mathrm{r}}_{x'} \exists d({}^{co}I(x') \wedge x = \frac{x'+d}{2}).$$

Since $x < -\left(\frac{1}{2}\right)^{n+2}$, we know that either $d = -1$ or $d = 0$.

$d = -1$: $x+1 = \frac{x'-1}{2}+1 = \frac{x'+1}{2}$ and from the inverse ${}^{co}I$-clause we get: ${}^{co}I(x+1)$.

$d = 0$: Substituting $x = \frac{x'}{2}$ into $x < -\left(\frac{1}{2}\right)^{n+2}$, gives $x' < -\left(\frac{1}{2}\right)^{n+1}$. Since x was arbitrary, the induction hypothesis also holds for x'. Hence, ${}^{co}I(x'+1)$. With $x+1 = \frac{x'}{2}+1 = \frac{(x'+1)+1}{2}$ the inverse ${}^{co}I$-clause gives us ${}^{co}I(x+1)$.

□

Lemma (`CoITimesTwo`)**.**

$$\forall^{\mathrm{nc}}_{x\in {}^{co}I}(-\frac{1}{2} < x < \frac{1}{2} \to {}^{co}I(2x)) \tag{19}$$

Proof. Let $x \in {}^{co}I$ with $-\frac{1}{2} < x < \frac{1}{2}$. Assume for instance $x = \frac{x_0+d_0}{2}$ for some $x_0 \in {}^{co}I$ and $|d_0| \leq 1$. We distinguish cases on d_0.

$d_0 = -1$: $-\frac{1}{2} < x < \frac{1}{2} \leftrightarrow -\frac{1}{2} < \frac{x_0-1}{2} < \frac{1}{2} \leftrightarrow 0 < x_0 < 2 \leftrightarrow -2 < -x_0 < 0$. From `CoIMinus` we have ${}^{co}I(-x_0)$. So `CoITimesTwoAux` gives ${}^{co}I(-x_0+1)$ and again `CoIMinus` ${}^{co}I(x_0-1)$. ${}^{co}I(2x)$ follows with $x_0 - 1 = 2\frac{x_0-1}{2} = 2x$.

$d_0 = 0$: Since $x_0 \in {}^{co}I$, $2x = 2\frac{x_0}{2} = x_0 = \frac{x'+d}{2}$ for some $x' \in {}^{co}I$ and $|d| \leq 1$. ${}^{co}I(2x)$ follows now from the inverse ${}^{co}I$-clause.

$d_0 = 1$: $-\frac{1}{2} < x < \frac{1}{2} \leftrightarrow -\frac{1}{2} < \frac{x_0+1}{2} < \frac{1}{2} \leftrightarrow -2 < x_0 < 0$. From ${}^{co}I(x_0)$ and `CoITimesTwoAux`, we get ${}^{co}I(x_0+1)$. ${}^{co}I(2x)$ follows with $x_0 + 1 = 2\frac{x_0+1}{2} = 2x$.

□

4 Formalisation

The informal proof from above is formalised in the interactive proof system Minlog. The formal proof sticks as closely as possible to the structure of the informal version. Occasionally, small properties need to be proven that are mostly computational irrelevant. They will go unmentioned here unless they highlight a particular aspect of the formal proof.

Since we want to extract a stream transformer, our goal formula

$$\forall^{\mathrm{nc}}_{x,y\in {}^{co}I}(-\frac{1}{4} < \frac{x\cdot y}{4} < \frac{1}{4} \to {}^{co}I(x\cdot y))$$

includes the non-computational quantifier $\forall^{\mathrm{nc}}_{x,y}$. The extracted program must not make computational use of x and y. Thus, the type of x, y is not predetermined. Indeed one might choose any implementation of the real numbers or even use a type variable for abstract real numbers. Any choice should lead to identical or in some sense equivalent programs.[2]

Using abstract real numbers can have the merit of easier proofs and simpler extracted terms, because for all non-computational parts of the proof the necessary properties can

[2]Here the concept of bisimilarity as briefly described in (Miyamoto, 2013, p. 38) can provide a suitable notion of equivalent programs.

just be assumed. The idea is that every implementation of real numbers will satisfy a minimal set of properties, i.e. the axioms of real numbers, and the properties derived from them. While this approach makes sense from the computational perspective of stream transformers, one should still strive for completeness and to show that all assumptions hold. This is why for the purpose of the thesis, I chose to focus on concrete real numbers; admittedly, without achieving the desired degree of completeness. In the following, I will explain my approach and give an account of all remaining global assumptions.

4.1 ...for concrete reals

Concrete reals

We view a real number as a pair of a Cauchy sequence of rationals and a modulus of type `rea`. Within this data type, we inductively define the predicate `Real` x, meaning that x is a proper real.

Definition. *A real number x is a pair $((a_n)_{n\in\mathbb{N}}, M)$ with $a_n \in \mathbb{Q}$ and $M : \mathbf{P}^3 \to \mathbb{N}$ such that $(a_n)_n$ is a Cauchy sequence with modulus M, that is*

$$|a_n - a_m| \leq 2^{-p} \quad \textit{for } n, m \geq M(p)$$

and M is weakly increasing. M is called Cauchy modulus of x.

Note that this definition draws on an existing construction of the binary numbers, the naturals, the integers and the rationals, which is provided to some completeness in the current Minlog version.

Real equality and comparison

Definition. *Two reals $x := ((a_n)_n, M)$, $y := ((b_n)_n, N)$ are called equal and written $x =^r y$, if*

$$|a_{M(p+1)} - b_{N(p+1)}| \leq 2^{-p} \quad \textit{for all } p \in \mathbf{P}.$$

This notion of equality (`RealEq`) comes quite naturally when one thinks of Cauchy sequences. We do not want to consider two reals only then equal, when they have the exact same Cauchy sequence and modulus (Leibniz-equality, in Minlog: `EqD`), but also

[3] $\mathbf{P}$ is the set of positive, binary numbers.

when their absolute difference converges in the above sense. A main part of this work was to find the correct way to deal with these different notions of equality and to adapt the definitions of the predicates I and ${}^{co}I$ accordingly.

Definition. *A real* $x := ((a_n)_n, M)$ *is called nonnegative (written* $x \in \mathbb{R}^{0+}$*) if*

$$-2^{-p} \leq a_{M(p)} \quad \textit{for all } p \in \mathbf{P}.$$

It is p-positive (written $x \in_p \mathbb{R}^+$*) if*

$$2^{-p} \leq a_{M(p+1)}.$$

Based on this definition, we can compare reals. We write $x \leq^r y$ for $y - x \in \mathbb{R}^{0+}$ and $x <_k y$ for $y - x \in_k \mathbb{R}^+$. Many basic properties of the comparison relations are still to be proven in the current Minlog system. As the ability to compare reals is essential to the proof of `CoITimesTwo`, I make use of some global assumptions. An account of them follows later.

A strategic decision is to avoid the strict comparison $<_p$, since the need to find a suitable p adds another dimension to the problem. Instead I use $x \leq^r y \to F$ for y strict less than x.

Program constants

The next step to implement concrete reals is to define the basic arithmetic functions (`RealPlus`, `RealUMinus`, `RealMinus`, `RealTimes`, `RealDiv`, `RealAbs`, `RealExp`, `RealMax`, `RealMin`) and show their properties. Of particular importance is the proof of compatibility with `RealEq`. For example: $\forall x, y, z, z_1(x =^r y \to z =^r z_1 \to x + z =^r y + z_1)$. This step is still work in progress in the current Minlog system.

For my main computations, I therefore introduce program constants. The idea is to avoid the use of real arithmetic functions (especially `RealDiv`) where possible and use their rational counterparts instead. The following program constants are used:

`RealAverage`	for $\frac{x+y}{2}$,
`RealAv`	for $\frac{x+d}{2}$,
`RealAvc`	for $\frac{x+y+i}{4}$,
`RealMult`	for $\frac{x \cdot y}{4}$, and
`RealMultc`	for $\frac{x \cdot y+z+i}{4}$.

In all cases they are assumed (a) to map proper reals to proper reals, e.g.

`RealAvReal`

$$\forall_{x,d}(\mathrm{Real}\ x \to \mathrm{Real}(\mathrm{RealAv}\ x\ d)), \tag{20}$$

(b) to map from $\mathbb{I}$ to $\mathbb{I}$, e.g.

`RealAvBd`

$$\forall_{x,d}(\mathrm{Real}\ x \to \mathrm{abs}\ x \leq^r 1 \to \mathrm{abs}\ d \leq 1 \to \mathrm{abs}(\mathrm{RealAv}\ x\ d) \leq^r 1), \tag{21}$$

and (c) to be compatible with real equality, e.g.

`RealAvCompat`

$$\forall_{x,y,d}(x =^r y \to \mathrm{RealAv}\ x\ d =^r \mathrm{RealAv}\ y\ d). \tag{22}$$

Occasionally as for `RealAvReal` this property is proven, or a proof is roughly sketched.

For the ordinary multiplication, I use the arithmetic function `RealTimes`.

The clauses for I and ${}^{co}I$

The above remarks result in adapted definitions of I and ${}^{co}I$. Minlog provides the feature to automatically add to an inductively defined predicate its "companion" predicate. Therefore, I only discuss the clause defining I with view on the resulting ${}^{co}I$-clause.

We define I by the clause

$$\forall^{\mathrm{nc}}_{x,y}\forall_d(\mathrm{Real}(y) \to |y| \leq^r 1 \to |d| \leq 1 \to I(y) \to I\left(\frac{y+d}{2}\right)) \tag{23}$$

and ${}^{co}I$ by the clause

$$\forall^{\mathrm{nc}}_x({}^{co}I(x) \to \exists^{\mathrm{r}}_y\exists_d(\mathrm{Real}(y) \land |y| \leq^r 1 \land |d| \leq 1 \land {}^{co}I(y) \land^l x =^r \frac{y+d}{2})). \tag{24}$$

This enriched definition deals with those subtleties that arise from our construction of the reals. By defining x, y and z as variables of the data type `rea`, one has to move the property that x is a proper real to the predicate `Real`. This fact is taken into consideration by the additional premise that each y generating a new element x of I must be a proper real.

The premise $|y| \leq^r 1$ ensures that I and ${}^{co}I$ stay within the unit interval $\mathbb{I}$. When adding a nullary constructor, this property could at least be proven for I by induction.

Since d is of type `int`, $|d| \leq 1$ takes care, that d can only be a signed digit $-1, 0$ or 1.

A consequence of these clauses is, that ${}^{co}I$ is closed under real equality:

Proposition (`CoIClosedRealEq`)**.**

$$\forall^{\mathrm{nc}}_x(\exists^{\mathrm{r}}_y(x =^r y \land^r {}^{co}I(y)) \to {}^{co}I(x))$$

Global Assumptions

In presenting the global assumptions, I will stick close to the notation used in Minlog to emphasise that they directly correspond to commands in Minlog.

I first make a couple of global assumptions that could later be included in the Minlog library or easily be proven from a more complete implementation of concrete reals.

`RealUMinusReal0`
$$\forall_x(\text{Real } x \to \text{Real } (^-x)) \tag{25}$$

`RealUMinusReal1`
$$\forall_x(\text{Real } (^-x) \to \text{Real } x) \tag{26}$$

`AbsUMinus`
$$\forall_x \text{abs } ^- x =^r \text{abs } x \tag{27}$$

`IntNOneTimesReal`
$$\forall_x(\text{IntN } 1 * x =^r {}^-x) \tag{28}$$

`ZeroTimesReal`
$$\forall_x(0 * x =^r 0_r) \tag{29}$$

`OneTimesReal`
$$\forall_x(1 * x =^r x) \tag{30}$$

`IntNOneCrit`
$$\forall_d(\text{abs } d \le 1 \to (0 \le d \to F) \to d = \text{IntN } 1) \tag{31}$$

`RealLeRefl`
$$\forall_x x =^r x \tag{32}$$

`RealLeTrans`
$$\forall_{x,y,z}(x \le^r y \to y \le^r z \to x \le^r z) \tag{33}$$

`RealLeAbsRatToLB`
$$\forall_{x,a}(\text{abs } x \le^r a \to {}^-a \le^r x) \tag{34}$$

`RealLeAbsRatToUB`
$$\forall_{x,a}(\text{abs } x =^r a \to x \le^r a) \tag{35}$$

`RealLeUMinusRatPosZero`
$$\forall_a(0 \le a \to {}^-a \le^r 0) \tag{36}$$

`RatLePosRatExpZero`
$$\forall_{d,a}(0 \le a \to 0 \le a^d) \tag{37}$$

`RealUMinusDistr`
$$\forall_{x,y} x + y =^r {}^-({}^-x + {}^-y) \tag{38}$$

`RealPlusZeroId`

$$\forall_x x + 0 =^r x \tag{39}$$

`RealLeRealUMinusZero`

$$\forall_x (0 \leq^r x \to {}^-x \leq^r 0) \tag{40}$$

`RealEqImplRealLe`

$$\forall_{x,y} (x =^r y \to x \leq^r y) \tag{41}$$

`RealLePlus0`

$$\forall_{x,a} (x \leq^r 0 \to x + a \leq^r a) \tag{42}$$

`RealLePlus1`

$$\forall_{x,a} (0 \leq^r x \to a \leq^r x + a) \tag{43}$$

`RatLeToRealLe`

$$\forall_{a,b} (a \leq b \to a \leq^r b) \tag{44}$$

`RealLeRatHalfTimesTwo1`

$$\forall_{x,a} (2 * x \leq^r a \to x \leq^r \text{RatHalf } a) \tag{45}$$

`RealLeRatHalfTimesTwo0`

$$\forall_{x,a} (a \leq^r 2 * x \to \text{RatHalf } a \leq^r x) \tag{46}$$

The following global assumptions concern the program constants `RealAverage`, `RealAv` and `RealAvc`.

`AverageRealAverage`

$$\begin{aligned} &\forall_{x,y,d,e} (\text{Real } x \to \text{Real } y \to \\ &\qquad \text{RealAverage}(\text{RealAv } x\ d)(\text{RealAv } y\ e) =^r \text{RealAvc } x\ y(d + e)) \end{aligned} \tag{47}$$

`RealLeRatHalfAv`

$$\forall_{k,y,d} (k \leq^r y \to 0 \leq d \to \text{RatHalf } k \leq^r \text{RealAv } y\ d) \tag{48}$$

`RealAvIntNOnePlusOne`

$$\forall_x \text{RealAv } x\ \text{IntN } 1 + 1 =^r \text{RealAv } x\ 1 \tag{49}$$

`RealAvZeroPlusOne`

$$\forall_x \text{RealAv } x\ 0 + 1 =^r \text{RealAv } (x+1)1 \tag{50}$$

`RealAvUMinus`

$$\forall_{x,d}(\text{Real } x \to \text{abs } d \leq 1 \to {}^-(\text{RealAv } x\ d) =^r \text{RealAv } {}^- x - d) \tag{51}$$

`RealLeZeroRealAvOne`

$$\forall_x(\text{abs } x \leq^r 1 \to 0 \leq^r \text{RealAv } x\ 1) \tag{52}$$

`TwoTimesRealAv`

$$\forall_{x,d}(2 * \text{RealAv } x\ d =^r x + d) \tag{53}$$

`RealLeRealAvRatHalf1`

$$\forall_{x,d,a}(x + d \leq^r a \to \text{RealAv } x\ d \leq^r \text{RatHalf } a) \tag{54}$$

`RealLeRealAvRatHalf2`

$$\forall_{x,d,a}(a \leq^r x + d \to \text{RatHalf } a \leq^r \text{RealAv } x\ d) \tag{55}$$

The following global assumptions concern the program constants J and K.

`JKAxiom`

$$\begin{aligned}&\forall_{x,y,d,e,i}(\text{Real } x \to \text{Real } y \to \\ &\text{RealAvc}(\text{RealAv } x\ d)(\text{RealAv } y\ e)i =^r \text{RealAv}(\text{RealAvc } x\ y(J\ d\ e\ i))(K\ d\ e\ i))\end{aligned} \tag{56}$$

`JBound`

$$\forall_{d,e,i}(\text{abs } d \leq 1 \to \text{abs } e \leq 1 \to \text{abs } i \leq 2 \to \text{abs}(J\ d\ e\ i) \leq 2) \tag{57}$$

`KBound`

$$\forall_{d,e,i}(\text{abs } d \leq 1 \to \text{abs } e \leq 1 \to \text{abs } i \leq 2 \to \text{abs}(K\ d\ e\ i) \leq 1) \tag{58}$$

(57) and (58) follow by case distinction and definition of J and K.

The following global assumptions concern the program constants `RealMult` and `RealMultc`.

`RealTimesRealMult`

$$\forall_{x,y}(\text{Real } x \to \text{Real } y \to x * y =^r 2 * (2 * \text{RealMult } x\ y)) \tag{59}$$

`RealMultRealMultc`

$$\forall_{x,y}\text{RealMult } x\ y =^r \text{RealMultc } x\ y\ 0\ 0 \tag{60}$$

The following global assumptions concern the program constants J_2 and K_2.

`JTwoBound`

$$\forall_{d,e_0,e,i}(\text{abs } d \leq 1 \to \text{abs } e_0 \leq 1 \to \text{abs } e \leq 1 \to \text{abs } i \leq 2 \to \text{abs}(J_2\ d\ e_0\ e\ i) \leq 2) \tag{61}$$

`KTwoBound`

$$\forall_{d,e_0,e,i}(\text{abs } d \le 1 \to \text{abs } e_0 \le 1 \to \text{abs } e \le 1 \to \text{abs } i \le 2 \to \text{abs}(K_2\ d\ e_0\ e\ i) \le 1) \quad (62)$$

(61) and (62) follow by case distinction and definition of J_2 and K_2.

`JTwoKTwoProp`

$$\begin{aligned}\forall_{x,x_1,y,z,z_2,z_3,z_4}\forall_{i,d_0,d,e_0,e}(x =^r \text{RealAv } x_1\ d \to z =^r \text{RealAv } z_2\ d_0 \to \\ \text{RealAvc } z_2(d * y)i =^r \text{RealAv } z_3\ e \to z_3 =^r \text{RealAv } z_4\ e_0 \to \\ \text{RealMultc } x\ y\ z\ i =^r \text{RealAv}(\text{RealMultc } x_1\ y\ z_4(J_2\ d\ e_0\ e\ i))(K_2\ d\ e_0\ e\ i))\end{aligned} \quad (63)$$

The assumption `JTwoKTwoProp` is needed in the proof of `CoIMultcSatCoICl` to solve the equation (14) and (15).

It remains one critical assumption. So far a constructive proof could not be found.

`RealLessThanZeroChar`

$$\forall_x((0 \le^r x \to \text{F}) \to \exists_n(-((\text{RatHalf } 1)^{n+1}) \le^r x \to \text{F})) \quad (64)$$

It turned out that (64) is actually computationally relevant for the proof of `CoITimesTwoAux`. It serves there the purpose to justify the use of the assertion

$$\forall_n\forall_x^{\text{nc}}((-\left(\frac{1}{2}\right)^{n+1} \le^r x \to F) \to {}^{co}I(x) \to {}^{co}I(x+1)),$$

which is shown by induction on n. However, since the proof of ${}^{co}I(x+1)$ in the assertion only uses the fact, that there is a n satisfying the first premise, the proof of `CoITimesTwoAux` should not rely on the particular n provided by (64). A possibly remedy is to drop the avoidance of $<_p$ in the premise and to use a suitably quantified p for the $<_p$-relation. For the time being, this issue remains unsolved.

4.2 ...for abstract reals

The basic difference of the abstract approach is to leave the type of the reals numbers open. For the formalisation, I use a type variable `r` to stand for real numbers. As one can not rely on given arithmetic functions a program constant has to be defined for every operation. Their basic properties are then assumed. A good approach is to keep the number of assumptions to a minimum.

The proof of (1) is only partially carried out, that is up to the proof of `CoIMulToCoI`. Since it yields no additional insight above that of the concrete formalisation, I will not further discuss this approach here. I want however point to its use to gain a first overview of the structure of a new argument as its more legitimate to leave many details open

to be solved by a concrete version. To clarify this point, note that the need to use the correct equality does not arise when a data type brings not along its own notion of equality.

5 Extraction

In this section, I present the extracted terms from the formalised proofs and explain how they operate. Whenever a proof uses coinduction the corresponding term contains a corecursion operator. For this reason I will start with discussing corecursion.

5.1 Corecursion

When computing infinite objects, such as streams, one needs corecursion. While recursion is applied to well-founded total ideals, works the constructor tree downwards and eventually terminates, corecursion is applied to and evaluates to cototal ideals. The corecursion operator ${}^{co}\mathcal{R}^{\tau}_{\mathbf{I}}$ for $\mathbf{I}$ is used to give a mapping from the type variable τ to the algebra $\mathbf{I}$ by corecursion on the structure of $\mathbf{I}$. It has the type

$$ {}^{co}\mathcal{R}^{\tau}_{\mathbf{I}} : \tau \to (\tau \to \mathbf{int} \times (\mathbf{I} + \tau)) \to \mathbf{I}. \tag{65} $$

The type $\mathbf{int} \times (\mathbf{I} + \tau)$ results from the definition of $\mathbf{I}$ by a single constructor C of type $\mathbf{int} \to \mathbf{I} \to \mathbf{I}$. The meaning of ${}^{co}\mathcal{R}^{\tau}_{\mathbf{I}} NM$ is given by the conversion rule

$$ {}^{co}\mathcal{R}^{\tau}_{\mathbf{I}} NM \mapsto C_{\pi_1(MN)}([\mathrm{id}^{\mathbf{I}\to\mathbf{I}}, \lambda_y({}^{co}\mathcal{R}^{\tau}_{\mathbf{I}} yM)]\pi_2(MN)), \tag{66} $$

where π_1, π_2 are the two projections of type $\rho \times \sigma$, and the notation $[f, g] : \rho + \sigma \to \tau$ (for $f : \rho \to \tau$ and $g : \sigma \to \tau$) is defined by

$$ [f, g](z) := \begin{cases} f(x) & \text{if } z = \mathrm{inl}(x), \\ g(y) & \text{if } z = \mathrm{inr}(y). \end{cases} $$

5.2 Minlog notation

Following (Berger et al., 2016, p. 33), I give here a short overview of the notational conventions of Minlog for the display of extracted terms:

Types	`iv`	base type for the algebra **I**
	`rho=>sigma`	function type
	`rho@@sigma`	product type
	`rho ysum sigma`	sum type

Variables (with fixed types)	`d,e`	of type **int**
	`v`	of type **I**
	`dv`	of type **int** $\times$ **I**
	`ivw`	of type **int** $\times$ **I** $\times$ **I**
	`jdvw`	of type **int** $\times$ **int** $\times$ **I** $\times$ **I**
	`iuvw`	of type **int** $\times$ **I** $\times$ **I** $\times$ **I**
	`iduvw`	0f type **int** $\times$ **int** $\times$ **I** $\times$ **I** $\times$ **I**

Constants	`Rec, CoRec`	recursion, corecursion
	`Des`	destructor
	`cL`	realiser for lemma L
	$\sim$	the overloaded unary minus

Terms	`[x]r`	lambda abstraction $\lambda_x r$
	`left r, right r`	components (prefix, binding strongest)
	`InL, InR`	injections into a sum type

5.3 The extracted terms

CoIClosedRealEq

The term extracted from `CoIClosedRealEq` as seen in figure 1 is of type `iv=>iv`. It takes `dv` as an argument and constructs an output with the constructor C using the left side of `dv` for the first argument and the right side of `dv` for the second. An "explicit mapping" is given by $dv \mapsto C_d v (= dv)$. So `cCoIClosedRealEq` is in some sense equivalent to the identity. We will therefore ignore later occurrences of `cCoIClosedRealEq` in other terms.

```
[dv]C left dv right dv
```

Figure 1: CoIClosedRealEq

CoIZero

The term extracted from `CoIZero` as seen in figure 2 is a constant function of type `iv`. The corecursion operator calls on `iv` with the step function `0@iv`. As desired it returns an infinite stream of $000\ldots$

```
(CoRec iv)(0@(DummyR iv))
```

Figure 2: CoIZero

CoIMultToCoIMultc

Figure 3 shows the term extracted from `CoIMultToCoIMultc`. `cCoIMultToCoIMultc` is a function of type `iv@iv=>int@iv@iv@iv`. It can be written as $(v, v_0) \mapsto (0_{\mathbf{int}}, v, v_0, 0_{\mathbf{I}})$ with $0_{\mathbf{int}} \in \mathbf{int}$ and $0_{\mathbf{I}}$ the zero stream in $\mathbf{I}$.

```
[v,v0]0@v@v0@cCoIZero
```

Figure 3: CoIMultToCoIMultc

CoIMinus

Figure 4 shows the extracted term of type `iv=>iv` of `CoIMinus`. The second argument of the corecursion operator, say $M : \mathbf{I} \to \mathbf{int} \times (\mathbf{I} + \mathbf{I})$ takes an argument v_0, destructs it into (d, v) and returns $M(d, v) = (-d, \mathrm{inr}(v))$. An implicit algorithm is $f : \mathbf{I} \to \mathbf{I}$:

$$f(C_d(v)) = (C_{-d}(f(v))).$$

Intuitively, speaking the job of this algorithm is to flip every signed digit to its additive inverse.

```
[v]
 (CoRec iv=>iv)v
 ([v0]~left Des v0@(InR iv iv)(cCoIClosedRealEq right Des v0))
```

Figure 4: CoIMinus

CoITimesDToCoI

The extracted term from `CoITimesDToCoI` as shown in figure 5 is a function of type `int=>iv=>iv` that evaluates according to the value of `cDCases: int=>uysum boole`. The term `cCoIClosedRealEq(cCoIMinus(cCoIClosedRealEq v))` applies if `k` satisfies the left hand side of the disjunction `k==IntN 1 orr k==0 oru k==1`, the term `cCoIClosedRealEq cCoIZero` if `k` satisfies the right and then the left side of the disjunction, and the term `cCoIClosedRealEq v` if `k` satisfies the right and then the right side of the disjunction. So the function is given by $\times_d : \mathbf{int} \to \mathbf{I} \to \mathbf{I}$:

$$\times_d(k, v) = \begin{cases} f(v) & \text{for } k = -1, \\ 0_{\mathbf{I}} & \text{for } k = 0, \\ v & \text{for } k = 1 \end{cases}$$

with f from `CoIMinus`.

```
[k,v]
 [if (cDCases k)
   (cCoIClosedRealEq(cCoIMinus(cCoIClosedRealEq v)))
   ([boole][if boole (cCoIClosedRealEq cCoIZero) (cCoIClosedRealEq v)])]
```

Figure 5: CoITimesDToCoI

CoIMultcSatCoIClause

The term extracted from `CoIMultcSatCoIClause` of type `int=>iv=>iv=>iv=>int@@ int@@iv@@iv@@iv` can be seen in figure 6. It contains the functions `Jprime` and `Kprime` of type `int=>int`. For $|k| \leq 6$ they satisfy the equation $k = J'(k) + 4K'(k)$ with $|J'(k)| \leq 2$ and $|K'(k)| \leq 1$. Introducing $+_{aux}$ for the function `cCoIAvcToCoI`, an implicit algorithm can directly be read off the term expression.

$$\begin{aligned} h(k, C_d(v'), v_0, C_{d_1}(v'_1)) \quad &\triangleq \text{let } C_e(C_{e_0}(w)) = +_{aux}(k, v'_1, \times_d(d, v_0)) \\ &= (J'(d + 2e + e_0 + k), K'(d + 2e + e_0 + k), v', v_0, w) \end{aligned}$$

```
[k,v,v0,v1]
 Jprime
 (left Des v+
  2*left Des(cCoIAvcToCoI(k@right Des v1@cCoITimesDToCoI left Des v v0))+
  left Des right Des(cCoIAvcToCoI
                     (k@right Des v1@cCoITimesDToCoI left Des v v0))+
  k)@
 Kprime
 (left Des v+
  2*left Des(cCoIAvcToCoI(k@right Des v1@cCoITimesDToCoI left Des v v0))+
  left Des right Des(cCoIAvcToCoI
                     (k@right Des v1@cCoITimesDToCoI left Des v v0))+
  k)@
 right Des v@
 v0@
 right Des right Des(cCoIAvcToCoI
                     (k@right Des v1@cCoITimesDToCoI left Des v v0))
```

Figure 6: CoIMultcSatCoIClause

CoIMultcToCoI

The extracted term of `CoIMultcToCoI` has type `int@@iv@@iv@@iv=>iv` and can be seen in figure 7. Consider the second argument of the corecursion operator, say $M : \mathbf{int} \times \mathbf{I} \times \mathbf{I} \times \mathbf{I} \to \mathbf{int} \times \mathbf{I} + \mathbf{int} \times \mathbf{I} \times \mathbf{I} \times \mathbf{I}$. With $N = (i, u, v, w)$ and h from `CoIMultcSatCoIClause` one can obtain

$$\begin{aligned} MN = M(i,u,v,w) \quad &\triangleq \text{ let } (i',d',u',v',w') = h(i,u,v,w) \\ &= (d', \mathrm{inr}(i',u',v',w')). \end{aligned}$$

Therefore by the conversion rule (66) one gets the implicit algorithm

$$\begin{aligned} \times_{aux}(i,u,v,w) = \quad &\triangleq \text{ let } (i',d',u',v',w') = h(i,u,v,w) \\ &= C_{d'}(\times_{aux}(i',u',v',w')). \end{aligned}$$

```
[iuvw]
 (CoRec int@@iv@@iv@@iv=>iv)iuvw
 ([iuvw0]
   [let iduvw
     (cCoIMultcSatCoIClause left iuvw0 left right iuvw0
     left right right iuvw0
     right right right iuvw0)
   (left right iduvw@(InR int@@iv@@iv@@iv iv)(left iduvw@right right iduvw))])
```

Figure 7: CoIMultcToCoI

CoIMultToCoI

Unsurprisingly, the extracted term of `CoIMultToCoI` as shown in figure 8 is the direct composition of `cCoIMultcToCoI` with `cCoIMultToMultc`.

```
[v,v0]cCoIClosedRealEq(cCoIMultcToCoI(cCoIMultToMultc v v0))
```

Figure 8: CoIMultToCoI

CoIShift

The extracted term (figure 9) of `CoIShift` has type `iv=>iv`. Consider the second argument of the corecursion operator, say $M : \text{int} \times \mathbf{I} \to \text{int} \times \mathbf{I} + \text{int} \times \mathbf{I}$. It takes arguments (d, v) and returns $(d, \text{inl}\,(v))$. Applied to $N = (1, v')$ with $v = C_{d'}(v')$ gives $MN = M(1, v') = (1, \text{inl}\,(v'))$. Hence the corecursion operator evaluates to $C_1(\text{id}^{\mathbf{I}\to\mathbf{I}}(v')) = C_1(v')$.

This is congruent with our intuition, that we just need to replace the first digit of a stream representation of x by 1, to find a stream representation of $x + 1$.

```
[v](CoRec int@@iv=>iv)(1@right Des v)([dv]left dv@(InL iv int@@iv)right dv)
```

Figure 9: CoIShift

CoITimesTwoAux

The extracted term of `CoITimesTwoAux` as seen in figure 10 is of type `iv=>iv`. As already stated in the discussion of the global assumption `RealLessThanZeroChar`, it would be

desirable to find a different proof or a slightly different formulation, that allowed to drop this presumably non-constructive assumption.

The proof of `CoITimesTwoAux` uses coinduction on n. In consequence the term contains the recursion operator. The idea is that the induction step does not make use of n. And indeed we find the step type to be `iv=>iv` instead of `n=>iv=>iv`.

Therefore, consider the inner term. For the argument $v = C_d(v')$, `cIntZeroIntNOneCrit` provides a case distinction for `d==IntN 1 oru d==Zero`. In case of $d = -1$ the first corecursion operator applies. Just the same as for `CoIShift`, it maps $v = C_{d'}(v') \mapsto C_1(v')$. In case of $d = 0$ the second corecursion operator applies. The first argument of the corecursion operator contains an additional function variable of type `iv=>iv`. The concrete function is given as an argument to the recursion operator. In this case it is `CoIShift`. So in the second case, one gets the mapping: $v = C_{d'}(v') \mapsto C_1(\mathrm{CoIShift}(v'))$.

Overall, an implicit function is given by

$$\times_2^{\mathrm{aux}}(C_d(v)) = \begin{cases} C_1(v') & \text{for } d = -1, \\ C_1(\mathrm{CoIShift}(v')) & \text{for } d = 0. \end{cases}$$

```
(Rec nat=>iv=>iv)cRealLessThanZeroChar cCoIShift
([n,(iv=>iv),v]
  [if (cIntZeroIntNOneCrit left Des v)
   ((CoRec int@@iv=>iv)(1@right Des v)([dv]left dv@(InL iv int@@iv)right dv))
    ((CoRec int@@iv=>iv)(1@(iv=>iv)right Des v)
    ([dv]left dv@(InL iv int@@iv)right dv))])
```

Figure 10: CoITimesTwoAux

CoITimesTwo

The term extracted from `CoITimesTwo` as shown in figure 11 is of type `iv=>iv`. For the argument $v = C_d(v')$, it distinguishes cases on d according to to corresponding paths in the disjunction `k==IntN 1 orr k==0 oru k==1`. One can read off the following function

$$\times_2(C_d(v)) = \begin{cases} f(\times_2^{\mathrm{aux}}(f(v'))) & \text{for } d = -1, \\ v' & \text{for } d = 0, \\ \times_2^{\mathrm{aux}}(v') & \text{for } d = 1. \end{cases}$$

with f the function from `UMinus`.

```
[v]
 [if (cDCases left Des v)
   (cCoIClosedRealEq
   (cCoIMinus
    (cCoIClosedRealEq
     (cCoITimesTwoAux(cCoIMinus(cCoIClosedRealEq right Des v))))))
   ([boole]
    [if boole
      (cCoIClosedRealEq right Des v)
      (cCoIClosedRealEq(cCoITimesTwoAux right Des v))])]
```

Figure 11: CoITimesTwo

Multiplication

As expected, the extracted term of `Multiplication` as shown in figure 12 makes direct use of the preliminary results, i.e. to transform two signed digit streams of x, y into the stream of the product $x \cdot y$, one needs to compute a stream for $\frac{x \cdot y}{4}$ by the program `cCoIMultToCoI` and then apply twice the procedure `cCoITimesTwo`.

```
[v,v0]cCoIClosedRealEq(cCoITimesTwo(cCoITimesTwo(cCoIMultToCoI v v0)))
```

Figure 12: Multiplication

6 Conclusion

I presented a formal proof of the existence of the product of two real numbers in [-1,1]. The formal proof used Cauchy sequences with modulus for the type of real numbers. This case study in constructive exact real arithmetic highlighted many aspects of the current implementation of constructive real numbers in Minlog, that demand future work. From the formal proof involving coinduction I extracted a term containing the corecursion operator. Thereby, I gave another example of the method of program extraction from proofs.

To highlight this aspect even more, the achieved results could be applied to a Haskell program allowing efficient multiplication of streams. Another path to pursue is the adaption of the methods and proof schemes, that are in action here, to obtain similar results for gray-code. Finally, as apparent in formalisation section, more work on the concrete implementation of the reals needs to be done.

References

U. Berger and M. Seisenberger. Proofs, programs, processes. In F. Ferreira, B. Löwe, E. Mayordomo, and L. M. Gomes, editors, *Proceedings CiE 2010*, volume 6158 of *LNCS*, pages 39–48. Springer-Verlag Berlin, Heidelberg, New York, 2010.

U. Berger, K. Miyamoto, H. Schwichtenberg, and H. Tsuiki. Logic for gray code computation. In D. Probst and P. Schuster, editors, *Concepts of Proof in Mathematics, Philosophy, and Computer Science*. De Gruyter, 2016.

A. Ciaffaglione and P. D. Gianantonio. A certified, corecursive implementation of exact real numbers. *Theoretical Computer Science*, 351:39–51, 2006.

K. Miyamoto. *Program extraction from coinductive proofs and its application to exact real arithmetic*. PhD thesis, Mathematisches Institut der Universität München, 2013.

K. Miyamoto and H. Schwichtenberg. Program extraction in exact real arithmetic. *Mathematical Structures in Computer Science*, 25(8):1692–1704, 2015.

H. Schwichtenberg and S. S. Wainer. *Proofs and Computations*. Association for Symbolic Logic and Cambridge University Press, 2012.

Eidesstattliche Erklärung

Hiermit erkläre ich, die vorliegende Arbeit selbständig verfasst zu haben und keine anderen als die in der Arbeit angegebenen Quellen und Hilfsmittel benutzt zu haben.

München, den 12.07.2016 Till Überrück-Fries