

LIMITS OF REAL NUMBERS IN THE BINARY SIGNED DIGIT REPRESENTATION

FRANZISKUS WIESNET^a AND NILS KÖPP^b

^a University of Trento, Via Sommarive 14, 38123 Povo
and Ludwig-Maximilians Universität, Theresienstr. 39, 80333 München
and TU Wien, Favoritenstraße 9-11, 1040 Wien

`franziskus.wiesnet@tuwien.ac.at`

^b Ludwig-Maximilians Universität, Theresienstr. 39, 80333 München
`koepp@math.lmu.de`

Published Aug. 19, 2022
(Submitted Mar. 30, 2021)

Abstract

We extract verified algorithms for exact real number computation from constructive proofs. To this end we use a coinductive representation of reals as streams of binary signed digits. The main objective of this paper is the formalisation of a constructive proof that real numbers are closed with respect to limits. All the proofs of the main theorem and the first application are implemented in the Minlog proof system and the extracted terms are further translated into Haskell. We compare two approaches. The first approach is a direct proof. In the second approach we make use of the representation of reals by a Cauchy-sequence of rationals. Utilizing translations between the two representation and using the completeness of the Cauchy-reals, the proof is very short.

In both cases we use Minlog’s program extraction mechanism to automatically extract a formally verified program that transforms a converging sequence of reals, i.e. a sequence of streams of binary signed digits together with a modulus of convergence, into the binary signed digit representation of its limit. The correctness of the extracted terms follows directly from the soundness theorem of program extraction.

As a first application we use the extracted algorithms together with Heron’s method to construct an algorithm that computes square roots with respect to the binary signed digit representation. In a second application we use the convergence theorem to show that the signed digit representation of real numbers is closed under multiplication.

Key words and phrases: signed digit code, exact real number computation, coinduction, corecursion, program extraction, realizability, Minlog, Haskell.

1 Introduction and motivation

1.1 Real numbers

Real numbers can be represented in several ways. One of the best-known representations is as Cauchy sequences of rational numbers together with a Cauchy modulus. Namely a *Cauchy real* is a pair $((a_n)_n, M)$ consisting of a sequence $(a_n)_n$ of real numbers and a modulus $M : \mathbb{Z}^+ \rightarrow \mathbb{N}$ such that $\forall_p \forall_{n,m > M(p)} |a_n - a_m| \leq 2^{-p}$, i.e. $(a_n)_n$ is a Cauchy sequence with modulus M .

However, in this paper the representation of real numbers as Cauchy reals will be just a tool. The main theorems of this paper are concerned with the signed digit representation of real numbers.

1.2 Binary representation vs. signed digit representation

The binary representation of a real number x in $[-1, 1]$ is given by

$$x = s \sum_{i=1}^{\infty} a_i 2^{-i},$$

where $s \in \{-1, 1\}$ and $a_i \in \{0, 1\}$ for every i . Here and further on, by equality = between two reals we mean an equivalence relation that is compatible with the usual operations and relations on the reals. In reality the specific of the real equality depends on the representation of real numbers. The binary representation of some concrete real number corresponds to a sequence of nested intervals. Reading the digits one after the other the interval is halved in each step. Hence from the binary code we can approximate a real number to arbitrary precision. Now consider the other direction, i.e. given a real number, compute the binary representation. This is not always possible, since the \leq -relation is not decidable. Further it is not possible to e.g. compute the binary representation of $\frac{x+y}{2}$ given representation of x and y . Here “compute” means that there is an algorithm which takes as input the binary streams of x and y and generates the binary stream representing $\frac{x+y}{2}$. In particular, the algorithm can only use finitely many binary digits of x and y in order to generate finitely many binary digits of $\frac{x+y}{2}$. For example, it is not possible to compute even the first digit (i.e. + or -) of the average of $+0??? \dots$ and $-0??? \dots$, where 0 is a list with entries 0 of arbitrary length and ? stands for an unknown digit. This is not possible due to the “gaps” in the binary representation. They are illustrated in Figure 1 at $0, \frac{1}{2}, -\frac{1}{2}, \frac{1}{4}$ and so on. From the first digit of a representation of a real x , we can decide $0 \leq x$ or $x \leq 0$, which in general can not be done if reasoning constructively about reals. The signed digit code fills these gaps. For a real number $x \in [-1, 1]$ it is defined by

$$x = \sum_{i=1}^{\infty} d_i 2^{-i},$$

where $d_i \in \{\bar{1}, 0, 1\}$ for every i . As the illustration in Figure 2 makes clear, to compute the first signed digit of a real number $x \in [-1, 1]$ we have to decide which of the cases $x \leq 0$, $-\frac{1}{2} \leq x \leq \frac{1}{2}$ or $0 \leq x$ holds. Now this is possible by application of the comparability theorem

$$\forall_{x,y,z} (x < y \rightarrow z \leq y \vee x \leq z).$$

Figure 2 also shows that the SD code of a real number (except -1 and 1) is not unique, whereas the binary code is “almost” everywhere unique. A stream of signed digits is an infinite list $d_1 d_2 d_3 \dots$ of elements in

$$\mathbf{Sd} := \{\bar{1}, 0, 1\}.$$

We will not use the signed digit streams directly, rather we use a coinductively defined predicate ${}^{\text{co}}\mathbf{I}$, which is given in the next section. For a real number x a realiser of $x \in {}^{\text{co}}\mathbf{I}$ is a signed digit stream representing x . The desired algorithms are given by the extracted terms of the proofs. The soundness theorem of program extraction gives correctness of these algorithms.

1.3 Historical background

One of the first papers where signed digits are used to represent real numbers, was published by Edwin Wiedmer in 1980 [Wie80]. The idea to use coinductive algorithms to describe the operators on the reals goes back to Alberto Ciaffaglione and Pietro Di Gianantonio [CG06] and was revised by Ulrich Beger and Tie Hou [BH08, Ber11]. The idea to use coinductively defined predicates together with the soundness theorem in this context is due to Ulrich Berger and Monika Seisenberger [BS12]. The notation and definitions in this paper are taken from [MS15] written by Kenji Miyamoto and Helmut Schwichtenberg. For the implementation of the translations between signed-digit and Cauchy-representation in Minlog see [Köp18].

1.4 Implementation in Minlog

For computing the extracted terms and verifying the correctness of the proofs, the proof assistant Minlog [Miy17] is used. An introduction to Minlog can be found in [Wie18] or `doc/tutor.pdf` in the Minlog directory. The implementation of the proofs can be found in the file `examples/analysis/sdlim.scm` in the directory of Minlog. After each proof we state its computational content not in the notation of Minlog but in the notation of Haskell, since the runtime of the programs in Haskell is shorter, and the terms can be defined in a more readable way. So after each proof we give the extracted term of the proof which was translated to Haskell using the command `terms-to-haskell-program`.

2 Formalisation

2.1 The theory of computational functionals (TCF)

We use the formal theory TCF to formalize statements like “ x is represented by some signed digit stream”. In this section we give a short overview of TCF. For a complete and formal introduction we refer to [SW12, Wie17].

In TCF all terms are typed. Types in TCF are either type variables, function types or algebras. Algebras can be seen as fixpoints of their constructors. For examples, the type \mathbb{N} of natural numbers is the algebra with the constructors $0 : \mathbb{N}$ and $S : \mathbb{N} \rightarrow \mathbb{N}$. In short notation we express this as $\mathbb{N} := \mu_{\xi}(\xi, \xi \rightarrow \xi)$, where μ is interpreted as least-fixed-point operator. Since each variable comes with a type, we will use the following naming conventions to suppress type declarations.

Notation 2.1. The following table shows which variables have which type.

$$\begin{array}{lll} m, n : \mathbb{N} & a, b : \mathbb{Q} & M, N : \mathbb{Z}^+ \rightarrow \mathbb{N} \\ d, e, k : \mathbb{Z} & x, y : \mathbb{R} & as, bs : \mathbb{N} \rightarrow \mathbb{Q} \\ p, q : \mathbb{Z}^+ & v, u : \mathbb{S} & xs, ys : \mathbb{N} \rightarrow \mathbb{R} \end{array}$$

If other variables are used, their type is either not relevant or we declare it individually.

Here, \mathbb{Z}^+ is defined as the positive (binary) numbers, \mathbb{Z} as the integers, \mathbb{Q} as the rational numbers and \mathbb{R} as real numbers. How these algebras are defined in detail however is not important for our purpose. In particular, in Minlog the type of real numbers \mathbb{R} is explicitly defined as the type $(\mathbb{N} \rightarrow \mathbb{Q}) \times (\mathbb{Z}^+ \rightarrow \mathbb{N})$. Since in the following proofs the concrete representation of real numbers is not important, we view \mathbb{R} as an abstract datatype and assume that we have abstract axiomatized reals with the usual operations including addition, multiplication, less-than and an equivalence-relation such that the other operations are compatible with it. We will refer to explicit representations by using predicates

e.g. $x \in \mathbf{R}$ and the computational content of a proof of this statement is a witness that x has an \mathbf{R} -representation.

In TCF predicates are defined (co-)inductively. Each inductively defined predicate comes with introduction axioms, also called clauses, and an elimination axiom. A coinductively defined predicate will be given by a closure axiom and a coinduction axiom. An example of an inductively defined predicate is the totality predicate \mathbf{T} which we discuss below or the predicate \mathbf{R} defined in Section 2.5. In Section 2.3 we introduce a coinductively defined predicate regarding the signed digit representation. This coinductively defined predicate is the main reason why TCF is the most suitable as underlying theory for our purpose. For an unary predicate A , we write $t \in A$ for At and $\forall_{t \in A} B$, $\exists_{t \in A} B$ are short for $\forall t (At \rightarrow B)$ and $\exists t (At \wedge B)$, respectively. Examples for these abbreviations that we later use include $x \in {}^{\text{co}}\mathbf{I}$, $x \in \mathbf{R}$, $d \in \mathbf{Sd}$.

Note that in TCF the existence quantifier as well as the conjunction and the distinction are formally inductively defined predicates. For examples, $A \wedge B$ is defined by the clause $A \rightarrow B \rightarrow A \wedge B$, in short notation $A \wedge B := \mu_X(A \rightarrow B \rightarrow X)$. As this notation suggest, $A \wedge B$ is the least predicate X which fulfills $A \rightarrow B \rightarrow X$. Furthermore, we have $A \vee B := \mu_X(A \rightarrow X, B \rightarrow X)$ and $\exists_t A := \mu_X(\forall_t (A \rightarrow X))$.

Another important property of TCF is that a term with a certain algebra as type does not have to consist of finitely many constructors of this type. For example, a natural number $n : \mathbb{N}$ does not have to be in the form $S \dots S0$. This means that terms in TCF are partial in general. E.g. we can also consider an infinite natural number which behaves like $SSS \dots$. However, we can not longer prove statements of the form $\forall_t A(t)$ by induction on t as we do not know how t is constructed and hence we can ad hoc not use something like induction on natural numbers. In order to use induction after all, we will use the totality predicate \mathbf{T} of TCF. Informally speaking, $t \in \mathbf{T}$ for some term $t : \tau$ means that t is a finite constructor expression if τ is an algebra, or t maps total object to total objects, if τ is a function type. E.g. for a sequence of natural numbers $ns : \mathbb{N} \rightarrow \mathbb{N}$ we have $ns \in \mathbf{T} := \forall_{n \in \mathbf{T}} (ns\ n) \in \mathbf{T}$ where $n \in \mathbf{T}$ is the inductive predicate given by the clauses $0 \in \mathbf{T}$ and $n \in \mathbf{T} \rightarrow (n + 1) \in \mathbf{T}$. The elimination axiom of $n \in \mathbf{T}$ is induction over natural numbers. Formally, the totality predicate is defined by recursion over the type. In particular, for each type we have an individual totality predicate. However, we do not mention this explicitly in the notation. For example, we just write $n \in \mathbf{T}$ instead of $n \in \mathbf{T}_{\mathbb{N}}$, as the type is clear from the context. We furthermore assume that predicates like \leq on natural numbers or (positive) integers are defined for total objects only. In particular, if we write something like $\forall_{n \geq M(p)} A$, we mean $\forall_n (n \in \mathbf{T} \wedge n \geq M(p) \rightarrow A)$.

2.2 Program extraction from proofs

In this section we give an overview on the process of program extraction from proofs in TCF. For formal definitions we refer to [Wie17, SW12, Sch21, Köp18]. In this short section we do not give formal definition as they are quite complex and we will use the proof assistant Minlog in any case to carry out the program extraction.

The computational content arises from the (co-)inductively defined predicates. When defining an (co-)inductively defined predicate or a predicate variable, it must also be determined whether it is computationally relevant (cr) or non computational (nc). For example, the totality predicate \mathbf{T} is defined as computationally relevant. The same goes for the predicates ${}^{\text{co}}\mathbf{I}$, \mathbf{R} and \mathbf{Sd} , which we introduce later. The equality and inequality on real numbers are non computational. A formula is computational relevant (cr), if its last conclusion is At where A is a computational relevant predicate. Note that the universal and existence quantifier by themselves will not carry computational content, in particular

the type of the formulas A and $\forall_x A$ are the same. But we will use the abbreviations $\forall_{t \in A}$ and $\exists_{t \in A}$, where $t \in A$ is computationally relevant (as long as A is). In the last section we said that we use abstract axiomatized real numbers. Here, we require that the axioms are non computational. This is the case for a usual axiomatisation as the axioms are about equations and inequalities.

In a first step, from a cr formula A the type *type* $\tau(A)$ and *realizer predicate* A^r are defined. Formally, this is done by recursion on the structure of the formula. The realiser predicate is a predicate which takes a term of the type of the formula and states that a term is a realizer of the formula, i.e. it adheres to the computational requirements stated in the formula.

In a second step, the *extracted term* $et(M)$ of the formalized proof M of A is computed. The extracted term is a λ -term with the type of the formula and is defined by recursion over proofs. It represents the extracted algorithm from the formal proof. In our case we will state this term after each proof which was formalized in Minlog, translated to the notation of Haskell.

In the last step of program extraction we generate a proof that the extracted term is indeed a realizer of the realizer predicate, i.e. $A^r et(M)$. This is the so-called soundness proof. Note that this proof can be generated automatically in Minlog.

In a nutshell, the result of formal program extraction is an algorithm in the form of a λ -term and the proof of its correctness. However, as it is hardly possible to describe a formal proof on paper, we use the proof assistant Minlog. In Minlog the last three steps above can be done automatically, so the laborious part is to find the right formulation of the theorem and the implementation of the constructive proof in Minlog. For the right formulation, we use the predicate ${}^{\text{co}}\mathbf{I}$ which is given in the next section.

2.3 Coinductive definition of the signed digit representation

Definition 2.1 (sd-code representation). *We define ${}^{\text{co}}\mathbf{I}$ as the greatest fixed point of the operator*

$$\Phi(X) := \left\{ x \mid \exists_{d \in \mathbf{Sd}, x'} \left(Xx' \wedge |x'| \leq 1 \wedge x = \frac{d + x'}{2} \right) \right\}.$$

A realiser of ${}^{\text{co}}\mathbf{I}x$ has the type

$$\tau({}^{\text{co}}\mathbf{I}) = \mu_{\tau(X)}(\tau(\Phi(X)) \rightarrow \tau(X)) = \mu_{\xi}(\mathbf{Sd} \rightarrow \xi \rightarrow \xi).$$

Here we have identified $\tau(\mathbf{Sd}) = \mu_{\xi}(\xi, \xi, \xi)$ with \mathbf{Sd} itself. We define $\mathbf{Str} := \tau({}^{\text{co}}\mathbf{I})$ and by \mathbf{C} we denote the only constructor of \mathbf{Str} . In Haskell notation \mathbf{Sd} and \mathbf{Str} are given by

```
data Sd = SdR | SdM | SdL
data Str = C Sd Str
```

In this notation we see that an element Cdv is a \mathbf{Sd} -stream with first digit d and tail v . Sometimes we abbreviate Cdv by just writing dv . We will also use this notation for reals: If we write something like dx for a real number x and a signed digit d , we mean $\frac{d+x}{2}$.

The definition of ${}^{\text{co}}\mathbf{I}$ as greatest fixpoint of Φ can be expressed by the two axioms

$${}^{\text{co}}\mathbf{I}^- : {}^{\text{co}}\mathbf{I} \subseteq \Phi({}^{\text{co}}\mathbf{I})$$

$${}^{\text{co}}\mathbf{I}^+ : X \subseteq \Phi({}^{\text{co}}\mathbf{I} \cup X) \rightarrow X \subseteq {}^{\text{co}}\mathbf{I},$$

where X is an unary predicate variable on real numbers. It is called *competitor predicate*. The first axiom ${}^{\text{co}}\mathbf{I}^-$ says that ${}^{\text{co}}\mathbf{I}$ is a fixpoint of Φ . Expressed in elementary formulas it

is given by

$$\forall_{x \in {}^{\text{co}}\mathbf{I}} \exists_{d \in \mathbf{Sd}, x'} \left(x' \in {}^{\text{co}}\mathbf{I} \wedge |x'| \leq 1 \wedge x = \frac{d + x'}{2} \right).$$

The type of this axiom is $\tau({}^{\text{co}}\mathbf{I}^-) = \mathbf{Str} \rightarrow \mathbf{Sd} \times \mathbf{Str}$ and a realiser is the destructor \mathcal{D} given by the computation rule

$$\mathcal{D}(Cdv) := \langle d, v \rangle.$$

The destructor takes a stream and returns a pair consisting of its first digit and its tail. Using the projectors π_0 and π_1 one gets the first digit and the tail, respectively. E.g. consider a cr formula of the form $x \in {}^{\text{co}}\mathbf{I} \rightarrow A$ of type $\mathbf{Str} \rightarrow \tau(A)$. Now assume that in its proof, ${}^{\text{co}}\mathbf{I}^-$ is used with $x \in {}^{\text{co}}\mathbf{I}$ at some point. On the computational level this corresponds to reading the head of the input stream and storing its tail.

The second axiom ${}^{\text{co}}\mathbf{I}^+$ expresses that ${}^{\text{co}}\mathbf{I}$ is the greatest fixpoint in a strong sense. It is explicitly given by:

$$\forall_x \left(Xx \rightarrow \forall_x \left(Xx \rightarrow \exists_{d \in \mathbf{Sd}, x'} \left(x' \in ({}^{\text{co}}\mathbf{I} \cup X) \wedge |x'| \leq 1 \wedge x = \frac{d + x'}{2} \right) \right) \rightarrow x \in {}^{\text{co}}\mathbf{I} \right) \quad (\star)$$

The type depends on the type of the predicate variable X , namely

$$\tau({}^{\text{co}}\mathbf{I}^+) = \tau(X) \rightarrow (\tau(X) \rightarrow \mathbf{Sd} \times (\mathbf{Str} + \tau(X))) \rightarrow \mathbf{Str}.$$

A realiser of ${}^{\text{co}}\mathbf{I}^+$ is the corecursion operator ${}^{\text{co}}\mathcal{R}$ which is given by the computation rule

$${}^{\text{co}}\mathcal{R}tf := \begin{cases} C(\pi_0(ft))v & \text{if } \pi_1(ft) = \text{in}_0(v) \\ C(\pi_0(ft)){}^{\text{co}}\mathcal{R}t'f & \text{if } \pi_1(ft) = \text{in}_1(t'). \end{cases}$$

Here in_0 and in_1 are the two constructors of the type sum $\mathbf{Str} + \tau(X)$. If $\pi_1(ft)$ has the form in_0v , the corecursion stops and we have $C(\pi_0(ft))v$ as signed digit representation. If it has the form in_1t' , the corecursion continues with the new argument t' . In both cases we have obtained at least the first digit $\pi_0(ft)$ of the stream. By iterating the corecursion we can generate each digit one by one. In Haskell we have

```
strDestr :: (Str -> (Sd, Str))
strDestr (C s u) = (s , u)

strCoRec :: (alpha -> ((alpha -> (Sd, (Either Str alpha))) -> Str))
strCoRec g h = (C (fst (h g))
  (case (snd (h g)) of
    { Left u0 -> u0 ;
      Right g1 -> (strCoRec g1 h) })))
```

Moreover we sometimes use the following functions.

```
hd :: (Str -> Sd)
hd u = fst(strDestr u)

tl :: (Str -> Str)
tl u = snd(strDestr u)

sdtoint :: Sd -> Integer
sdtoint SdR = 1
sdtoint SdM = 0
sdtoint SdL = -1

id :: Pos -> Nat
id p = p
```

Here `fst` and `snd` are the pair-projections.

2.4 Basic lemmas

We prove two basic lemmas, which will often occur in the proofs following:

Lemma 2.2 (CoICompat). *The predicate ${}^{\text{co}}\mathbf{I}$ is compatible with real equality, i.e.*

$$\forall_{x \in {}^{\text{co}}\mathbf{I}} \forall_y (x = y \rightarrow y \in {}^{\text{co}}\mathbf{I})$$

Proof. We apply (\star) to the predicate $Px := \exists_{y \in {}^{\text{co}}\mathbf{I}} (x = y)$:

$$\forall_x \left(Px \rightarrow \forall_x \left(Px \rightarrow \exists_{d \in \mathbf{Sd}, x'} \left(x' \in ({}^{\text{co}}\mathbf{I} \cup P) \wedge |x'| \leq 1 \wedge x = \frac{d + x'}{2} \right) \right) \rightarrow x \in {}^{\text{co}}\mathbf{I} \right)$$

It is sufficient to prove the second premise. So assume x and $y \in {}^{\text{co}}\mathbf{I}$ with $x = y$ are given. Using $y \in {}^{\text{co}}\mathbf{I}$ with ${}^{\text{co}}\mathbf{I}^-$ we get $e \in \mathbf{Sd}$ and $y' \in {}^{\text{co}}\mathbf{I}$ with $|y'| \leq 1$ and $y = \frac{e + y'}{2}$. Hence $d := e$ and $x' := y'$ have the desired properties. \square

In the following proofs, this theorem is used tacitly. In Minlog it has the name `CoICompat`. The extracted term of this theorem is given by

```
cCoICompat :: (Str -> Str)
cCoICompat u0 = strCoRec u0 (\ su1 -> (case su1 of
  {s2 u2 -> (s2, Left u2)}))
```

Assume \mathbf{f} is the costep-function above, then for $\mathbf{C} \ \mathbf{s} \ \mathbf{u}$ some stream we have $\mathbf{f}(\mathbf{C} \ \mathbf{s} \ \mathbf{u}) = (\mathbf{s}, \text{Left} \ \mathbf{u})$. So if we unfold the `strCoRec` we get

```
cCoICompat (C s u) = C s u,
```

i.e. the computational content of this lemma is actually the identity. Hence to increase readability, we will leave it out in the following.

Lemma 2.3 (CoIClosureInv).

$$\forall_{x \in {}^{\text{co}}\mathbf{I}, d \in \mathbf{Sd}} \frac{d + x}{2} \in {}^{\text{co}}\mathbf{I}$$

Proof. We use (\star) with the predicate

$$Px := \exists_{d \in \mathbf{Sd}, x' \in {}^{\text{co}}\mathbf{I}} x = \frac{d + x'}{2},$$

Again, in order to prove the goal formula, it is sufficient to prove the second premise. Therefore our goal is

$$\forall_x \left(\exists_{d \in \mathbf{Sd}, x' \in {}^{\text{co}}\mathbf{I}} \left(x = \frac{d + x'}{2} \right) \rightarrow \exists_{d \in \mathbf{Sd}, x'} \left(x' \in ({}^{\text{co}}\mathbf{I} \cup P) \wedge |x'| \leq 1 \wedge x = \frac{d + x'}{2} \right) \right).$$

But this follows immediately from ${}^{\text{co}}\mathbf{I} \subseteq {}^{\text{co}}\mathbf{I} \cup P$ and $x \in {}^{\text{co}}\mathbf{I} \rightarrow |x'| \leq 1$. \square

In Minlog this lemma has the name `CoIClosureInv` and its extracted term is given by

```
cCoIClosureInv :: (Sd -> (Str -> Str))
cCoIClosureInv s0 u1 = strCoRec (s0 , u1)
  (\ su2 -> (
    (case su2 of
      { (,) s u -> s } ) ,
    (Left (case su2 of
      { (,) s0 u0 -> u0 }))))),
```

which is an elaborate way to write the constructor \mathbf{C} , namely if \mathbf{f} is the costep-function above then $\mathbf{f} \text{ (s0,u1) } = (\text{s0,Left u1})$ and by unfolding `strCoRec cCoIClosureInv s0 u1 = C s0 u1`.

2.5 Cauchy reals and signed digit streams

We now formalize the relation between reals represented by Cauchy-sequences of rationals and signed digit streams.

Definition 2.4 (Cauchy representation). *We denote $as : \mathbb{N} \rightarrow \mathbb{Q}$ and $M : \mathbb{Z}^+ \rightarrow \mathbb{N}$ and define*

$$\mathbf{Mon}(M) := M \in \mathbf{T} \wedge \forall_{p \leq q} (Mp \leq Mq).$$

For a real x we write $x \in \mathbf{R}$, if there exist $M \in \mathbf{Mon}$ and $as \in \mathbf{T}$ with

$$\forall_{p \in \mathbf{T}} \forall_{n \geq M(p)} \left(|x - (as \ n)| < 2^{-(p+1)} \right),$$

i.e. there is a sequence of rationals converging to x .

In Haskell this representation is given by the datatype

```
data Rea = RealConstr (Nat -> Rational) (Pos -> Nat),
```

with the pair-projections

```
realSeq :: (Rea -> (Nat -> Rational))
realSeq (RealConstr as m) = as

realMod :: (Rea -> (Pos -> Nat))
realMod (RealConstr as m) = m.
```

In the following we will prove that for some $x \in {}^{\text{co}}\mathbf{I}$ and $n \in \mathbf{T}$ there exists a rational approximation to x with precision $\frac{1}{2^n}$. To get a sequence of rationals representing x we need dependent choice. Moreover later we will use countable choice.

Definition 2.5 (Choice Principles). *We denote $f : \mathbb{N} \rightarrow \alpha$, then the axiom of dependent choice \mathbf{DC} is given by*

$$\exists_{\alpha} P(0, \alpha) \rightarrow \forall_{n \in \mathbf{T}, \alpha} (P(n, \alpha) \rightarrow \exists_{\alpha} P(n+1, \alpha)) \rightarrow \exists_f \forall_{n \in \mathbf{T}} P(n, fn).$$

It has the type

$$\tau(P) \rightarrow (\mathbb{N} \rightarrow \tau(P) \rightarrow \tau(P)) \rightarrow \mathbb{N} \rightarrow \tau(P)$$

and the realizer is given by the recursion operator for \mathbb{N} , i.e.

```
natRec :: Nat -> a -> (Nat -> a -> a) -> a
natRec 0 g h = g
natRec n g h | n > 0 = h (n - 1) (natRec (n - 1) g h).
```

The axiom of countable choice \mathbf{CC} is given by

$$\forall_{n \in \mathbf{T}} \exists_{\alpha \in \mathbf{T}} P(n, \alpha) \rightarrow \exists_{f \in \mathbf{T}} \forall_{n \in \mathbf{T}} P(n, fn)$$

Its type is $(\mathbb{N} \rightarrow \alpha \times \tau(P)) \rightarrow (\mathbb{N} \rightarrow \alpha) \times (\mathbb{N} \rightarrow \tau(P))$ and the realizer is basically given by the identity, namely

$$\lambda F \langle \lambda n (Fn)_0, \lambda n (Fn)_1 \rangle.$$

Theorem 2.6 (StrToCs). *Any real x represented by a signed digit code can be represented by a Cauchy-sequence, i.e.*

$$\forall_{x \in {}^{\text{co}}\mathbf{I}} (x \in \mathbf{R} \wedge |x| \leq 1).$$

Proof. Assume $x \in {}^{\text{co}}\mathbf{I}$, then $|x| \leq 1$ holds by ${}^{\text{co}}\mathbf{I}^-$. Now let

$$P(n, a) := a \in \mathbf{T} \wedge \exists_{y \in {}^{\text{co}}\mathbf{I}} (x = 2^{-(n+1)}y + a)$$

We want to apply **DC**, hence we first prove $\exists_a P(0, a)$. By ${}^{\text{co}}\mathbf{I}^-$ there is $y \in {}^{\text{co}}\mathbf{I}, d \in \mathbf{Sd}$ with $x = \frac{y+d}{2}$, so let $a := \frac{d}{2}$. For the second premise of **DC** assume $n \in \mathbf{T}, a \in \mathbf{T}$ and $P(n, a)$ i.e. there exists $y \in {}^{\text{co}}\mathbf{I}$ with $x = \frac{1}{2^{n+1}}y + a$. We need to prove

$$\exists_b (b \in \mathbf{T} \wedge \exists_{z \in {}^{\text{co}}\mathbf{I}} (x = 2^{-(n+2)}z + b))$$

Since $y \in {}^{\text{co}}\mathbf{I}$ we get $d \in \mathbf{Sd}, y' \in {}^{\text{co}}\mathbf{I}$ with $y = \frac{y'+d}{2}$. Now we choose $b := a + \frac{d}{2^{n+2}}$ and $z = y'$, then

$$x = \frac{1}{2^{n+1}}y + a = \frac{1}{2^{n+2}}y' + b$$

Hence by **DC** there exists $as \in \mathbf{T}$ with

$$\forall_{n \in \mathbf{T}} \exists_{y \in {}^{\text{co}}\mathbf{I}} (x = 2^{-(n+1)}y + (as \ n)).$$

Hence with $Mp := p$ and $|y| \leq 1$ we get $x \in \mathbf{R}$. □

The extracted term of the proof is given by

```

cStrToCsInit :: (Str -> (Rational, Str))
cStrToCsInit u0 = (sdtoint(hd u0) % 2 , tl u0)

cStrToCsStep :: (Nat -> ((Rational, Str) -> (Rational, Str)))
cStrToCsStep n0 (a,u0) = (a + sdtoint(hd u0) % (((2 ^ n0) * 2) * 2)
,
  tl u0)

cStrToCs :: (Str -> Rea)
cStrToCs u0 = (\ n1 -> (fst (natRec n1 (cStrToCsInit u0)
  cStrToCsStep)) ,
  id) .

```

Here `cStrToCsInit` corresponds to the first premise and `cStrToCsStep` to the second premise of **DC** in the proof. **DC** itself only appears as the `natRec` term in `cStrToCs`. Informally we can represent the computational content by

$$d_0 d_1 d_2 \cdots \mapsto \left(\left(\sum_{i=1}^n d_i 2^{-i} \right)_n, \iota \right),$$

where $\iota : \mathbb{Z}^+ \rightarrow \mathbb{N}$ is the canonical inclusion. For the converse of Theorem 2.7 we first prove the following two lemmas.

Lemma 2.7 (Special case of ApproxSplit). *Let $a, b : \mathbb{Q}$ then*

$$\forall_{a,b \in \mathbf{T}, x \in \mathbf{R}} (a < b \rightarrow a \leq x \vee x \leq b).$$

Proof. Given (as, M) a Cauchy-sequence converging to x . Find p be such that $\frac{1}{2^p} < b - a$ (which is possible since $a, b \in \mathbf{T}$). Then for $n \geq Mp$

$$|x - (as\ n)| \leq 2^{-(p+1)},$$

i.e. $x \in [(as\ n) - \frac{1}{2^{p+1}}, (as\ n) + \frac{1}{2^{p+1}}]$. Case $(as\ n) \leq \frac{a+b}{2}$. In that case $x \leq (as\ n) + \frac{1}{2^{p+1}} < b$. Otherwise $a \leq x$. \square

Note that it can be proven more generally for $y, z \in \mathbf{R}$ and $y < z$ instead of a, b , but we will actually only need it for the special cases $0 < \frac{1}{2}$ and $-\frac{1}{2} < 0$. The extracted term for the former case is:

```
cApproxSplitZeroPtFive :: (Rea -> Bool)
cApproxSplitZeroPtFive (RealConstr as m) = (as (m 3)) <= (1/4)
```

where `cRatLeAbsBound` is the extracted term of a proof of $\forall_{a \in \mathbf{T}} \exists_{n \in \mathbf{T}} |a| \leq 2^n$. For the converse of Theorem 2.7 we first prove the following lemma.

Lemma 2.8 (CsToStrAux). *For all $x \in \mathbf{R}$ with $|x| \leq 1$*

$$\exists_{d \in \mathbf{Sd}, y \in \mathbf{R}} \left(|y| \leq 1 \wedge x = \frac{y + d}{2} \right).$$

Proof. Let (as, M) a Cauchy-sequence converging to x . We use Lemma 2.8 with $0 < \frac{1}{2}$ respectively $-\frac{1}{2} < 0$. We define

$$d := \begin{cases} \bar{1} & \text{if } x < 0, \\ 0 & \text{if } -\frac{1}{2} < x < \frac{1}{2}, \\ 1 & \text{if } 0 < x, \end{cases}$$

$bs\ n := 2(as\ n) - d$, $Np := M(p + 1)$ and $y = 2x - d$. Then (bs, N) is a Cauchy-sequence converging to y . Furthermore $x = \frac{1}{2}(y + d)$ and $|y| \leq 1$ by definition. \square

The extracted term is given by

```
cCsToStrAux :: (Rea -> (Sd, Rea))
cCsToStrAux (RealConstr as m) =
  (if ((as (m 3)) <= -1/4) then
    (SdL , (RealConstr (\ n3 -> (((2) * (as n3)) + (1)))
      (\ p3 -> (m (p3 + 1)))))
  else
    (if ((as (m 3)) <= 1/4) then
      (SdM , (RealConstr (\ n3 -> ((2) * (as n3)))
        (\ p3 -> (m (p3 + 1)))))
    else
      (SdR , (RealConstr (\ n3 -> (((2) * (as n3)) + (-1)))
        (\ p3 -> (m (p3 + 1)))))))).
```

Again we can represent the computational content informally, namely

$$(as, M) \mapsto (g(as, M), (h(as, M), N)),$$

where $Np := M(p + 1)$ and g, h are the functions

$$g(as, M) := \begin{cases} \bar{1} & \text{if } as(M\ 3) \leq -\frac{1}{4}, \\ 0 & \text{if } |as(M\ 3)| \leq \frac{1}{4}, \\ 1 & \text{otherwise.} \end{cases}$$

$$h(as, M) := 2as - g(as, M)$$

Using this lemma the proof of the translation from Cauchy-sequences to stream is very short:

Theorem 2.9 (CsToStr).

$$\forall_x (x \in \mathbf{R} \rightarrow |x| \leq 1 \rightarrow x \in {}^{\text{co}}\mathbf{I})$$

Proof. Assume $x \in \mathbf{R}$ and $|x| \leq 1$. We use (\star) with

$$Px := \exists_{d \in \mathbf{Sd}, y \in \mathbf{R}} \left(|y| \leq 1 \wedge x = \frac{y+d}{2} \right).$$

By the previous lemma it suffices to prove the second premise, namely

$$\forall_x \left(Px \rightarrow \exists_{d \in \mathbf{Sd}, x'} \left(x' \in ({}^{\text{co}}\mathbf{I} \cup P) \wedge |x'| \leq 1 \wedge x = \frac{d+x'}{2} \right) \right).$$

But this follows immediately by another application of the lemma, namely let $d \in \mathbf{Sd}, y \in \mathbf{R}$ with $|y| \leq 1$ and $x = \frac{y+d}{2}$. Then by the lemma there are $e \in \mathbf{Sd}, z \in \mathbf{R}$ with $|z| \leq 1$ and $y = \frac{z+e}{2}$ and hence $y \in P \subseteq {}^{\text{co}}\mathbf{I} \cup P$. \square

The extracted term is

```

cCsToStr :: (Rea -> Str)
cCsToStr x0 = strCoRec
  (cCsToStrAux x0)
  (\ sx1 -> (case sx1 of { (,) s0 x0 ->
    (case (cStrToCsAux x0) of { (,) s1 x1 ->
      (s0 , (Right (s1 , x1))) } } ) ) )

```

and informally

$$x \mapsto \pi_0(\text{cCsToStrAux } x) :: \text{cCsToStr}(\pi_1(\text{cCsToStrAux } x)).$$

3 Convergence theorem

The convergence theorem states that the signed digit representation is closed under limits. In this section we consider a direct proof of this theorem, which only relies on the signed digit representation of real numbers, and an indirect proof, which works with Cauchy reals and uses the translation between the signed digit code and Cauchy reals. After proving the convergence theorem in these two ways, we compare the extracted terms of both proofs.

Definition 3.1 (Convergence). *Let $xs : \mathbb{N} \rightarrow \mathbb{R}$ and $M \in \mathbf{Mon}$ then xs is a Cauchy-sequence with modulus M iff*

$$\forall_{p \in \mathbf{T}} \forall_{n, m \geq M(p)} |(xs \ n) - (xs \ m)| \leq 2^{-p},$$

we also write $\mathbf{Cauchy}(xs, M)$. The sequence xs converges to x with Modulus M iff

$$\forall_{p \in \mathbf{T}} \forall_{n \geq M(p)} |x - (xs \ n)| \leq 2^{-p},$$

we write $\mathbf{Conv}(xs, x, M)$.

The convergence theorem can now be stated in the following way.

Theorem 3.2 (SdLim). *Let $xs : \mathbb{N} \rightarrow \mathbb{R}$ be a sequence of reals in ${}^{\text{co}}\mathbf{I}$ which converges to some real x with modulus M . Then $x \in {}^{\text{co}}\mathbf{I}$, i.e.*

$$\forall_{x, xs; M \in \mathbf{Mon}} (\forall_{n \in \mathbf{T}} (xs \ n) \in {}^{\text{co}}\mathbf{I} \rightarrow \mathbf{Conv}(xs, x, M) \rightarrow x \in {}^{\text{co}}\mathbf{I}).$$

3.1 Direct approach

The following approach was already considered in [Wie21] and is adjusted here to our setting.

Lemma 3.3 (CoINegToCoIPlusOne, CoIPosToCoIMinusOne).

$$\begin{aligned} \forall_{x \in {}^{\text{co}}\mathbf{I}} (x \leq 0 \rightarrow {}^{\text{co}}\mathbf{I}(x + 1)) \\ \forall_{x \in {}^{\text{co}}\mathbf{I}} (0 \leq x \rightarrow {}^{\text{co}}\mathbf{I}(x - 1)) \end{aligned}$$

Proof. Since the proofs are very similar, we only prove the first formula. We use (\star) with $Px := \exists_{y \in {}^{\text{co}}\mathbf{I}} (y \leq 0 \wedge y + 1 = x)$. We need to prove the second premise, namely

$$\forall_x \left(Px \rightarrow \exists_{d \in \mathbf{Sd}, x'} \left(x' \in ({}^{\text{co}}\mathbf{I} \cup P) \wedge |x'| \leq 1 \wedge x = \frac{d + x'}{2} \right) \right)$$

Let $x \in P, y \in {}^{\text{co}}\mathbf{I}$ with $y \leq 0$ and $y + 1 = x$ be given. Our goal is

$$\exists_{d \in \mathbf{Sd}, x'} \left(x' \in ({}^{\text{co}}\mathbf{I} \cup P) \wedge |x'| \leq 1 \wedge x = \frac{d + x'}{2} \right).$$

From $y \in {}^{\text{co}}\mathbf{I}$ we get e and y' with $e \in \mathbf{Sd}, y' \in {}^{\text{co}}\mathbf{I}, |y'| \leq 1$ and $y = \frac{e + y'}{2}$. We make a case distinction on $\mathbf{Sd} \ e$: If $e = -1$, we define $d := 1 \in \mathbf{Sd}$ and $x' := y'$. Then $|x'| \leq 1$ and $x' \in {}^{\text{co}}\mathbf{I}$ by definition. Furthermore we have

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

If $e = 0$, we define $d := 1$ and $x' := y' + 1$. In this case we prove Px' , namely we show $y' \in {}^{\text{co}}\mathbf{I}, y' \leq 0$ and $x' = y' + 1$. We only need to prove $y' \leq 0$ which follows directly from $y \leq 0$ and $y = \frac{0 + y'}{2}$. The last case is $e = 1$. Because of $y \leq 0, y = \frac{-1 + y'}{2}$ and $|y'| \leq 1$, this is only possible if y is equal to 0, and therefore $x = 1$. Hence we define $d := 1$ and $x' := 1$. Then $x = \frac{d + x'}{2}$ and $x' = 1 \in {}^{\text{co}}\mathbf{I}$ is easily proven by coinduction. (For details we refer to the Minlog implementation of the theorem `CoIOne` in `examples/analysis/sddiv.scm`.) \square

A realizer of the first formula is a function \mathbf{f} , which takes a signed digit stream of a real number x and returns a signed digit stream of $x + 1$ if $x \leq 0$. The extracted term of the proof of the first statement translated to Haskell is

```
cCoINegToCoIPlusOne :: (Str -> Str)
cCoINegToCoIPlusOne u0 = aiCoRec u0
  (\ u1 -> (case (hd u1) of
    { SdR -> (SdR , (Left cCoIOne)) ;
      SdM -> (SdR , (Right (tl u1))) ;
      SdL -> (SdR , (Left (tl u1))) })))
```

where

```
cCoIOne :: Str
cCoIOne = (aiCoRec () (\ g -> (SdR , (Right ())))))
```

is the stream representing 1. Unfolding `strCoRec` once yields `cCoIOne = C SdR cCCoIOne`, i.e. it is a constant stream of `SdR`. Another way to characterise this function \mathbf{f} is to give its computation rules:

$$\begin{aligned} \mathbf{f}(C\bar{1}v) &:= C1v \\ \mathbf{f}(C0v) &:= C1(\mathbf{f}v) \\ \mathbf{f}(C1v) &:= [1, 1, \dots] \end{aligned}$$

Analogously as extracted term of the second statement of this lemma, we get a function $g : \mathbf{Str} \rightarrow \mathbf{Str}$ which is characterised by the rules

$$\begin{aligned} g(C\bar{1}v) &:= [\bar{1}, \bar{1}, \dots] \\ g(C0v) &:= C\bar{1}(gv) \\ g(C1v) &:= C\bar{1}v. \end{aligned}$$

It takes a signed digit stream of a real x and returns a signed digit stream of $x - 1$ if $0 \leq x$. Using this lemma, we are now able to prove the following lemma:

Lemma 3.4 (CoIToCoIDouble).

$$\forall_{x \in {}^{\text{co}}\mathbf{I}} \left(|x| \leq \frac{1}{2} \rightarrow 2x \in {}^{\text{co}}\mathbf{I} \right)$$

Proof. We apply ${}^{\text{co}}\mathbf{I}^-$ and get $d \in \mathbf{Sd}$, $x' \in {}^{\text{co}}\mathbf{I}$ with $|x'| \leq 1$ and $x = \frac{d+x'}{2}$. We distinguish cases on $d \in \mathbf{Sd}$: $d = 1$: Then $2x - 1 = x'$ and $|x| \leq \frac{1}{2}$ which imply $x' \leq 0$. By the first part of Lemma 3.3 $1 + x' = 2x \in {}^{\text{co}}\mathbf{I}$. $d = -1$: As the first case but with the second part of Lemma 3.3. $d = 0$: In this case $2x = x'$ and $x' \in {}^{\text{co}}\mathbf{I}$ by assumption. \square

In Haskell notation the extracted term is given by

```
cCoIToCoIDouble :: (Str -> Str)
cCoIToCoIDouble u0 = case (hd u0) of
  { SdR -> (cCoINegToCoIPlusOne (tl u0)) ;
    SdM -> (tl u0) ;
    SdL -> (cCoIPosToCoIMinusOne (tl u0)) }
```

Again we give a more readable characterisation of the extracted term D by the computation rules

$$\begin{aligned} D(C\bar{1}u) &:= gu \\ D(C0u) &:= u \\ D(C1u) &:= fu, \end{aligned}$$

where f, g are the computational content of the previous lemma. The following lemma is a special case of

$$\forall_{x \in {}^{\text{co}}\mathbf{I}, y \in {}^{\text{co}}\mathbf{I}} \frac{x + y}{2} \in {}^{\text{co}}\mathbf{I}.$$

This theorem is implemented as the theorem **Average** in `examples/analysis/average.scm` of Minlog and was considered in [BS12, MS15]. But here we give a direct proof of a special case because it is instructive and elementary.

Lemma 3.5 (Special case of CoIAverage).

$$\forall_{x \in {}^{\text{co}}\mathbf{I}} \left(\frac{x}{2} \pm \frac{1}{4} \right) \in {}^{\text{co}}\mathbf{I}$$

Proof. Applying ${}^{\text{co}}\mathbf{I}^-$ yields $x' \in {}^{\text{co}}\mathbf{I}$ and $d \in \mathbf{Sd}$ with $x = \frac{d+x'}{2}$. We show only ${}^{\text{co}}\mathbf{I}(\frac{x}{2} + \frac{1}{4})$, the other case is proven analogously. We distinguish cases on $d \in \mathbf{Sd}$: $d = 1$: Then $\frac{x}{2} + \frac{1}{4} = \frac{2+x'}{4} = \frac{1}{2}(1 + \frac{x'}{2})$. $d = 0$: Then $\frac{x}{2} + \frac{1}{4} = \frac{1+x'}{4} = \frac{1}{2}\frac{1+x'}{2}$. $d = -1$: Then $\frac{x}{2} + \frac{1}{4} = \frac{x'}{4} = \frac{1}{2}\frac{x'}{2}$. In each case we apply Lemma 2.4 twice to get $(\frac{x}{2} + \frac{1}{4}) \in {}^{\text{co}}\mathbf{I}$. \square

We denote the extracted term of the proven statement by q^+ . From the proof and the fact that the extracted term of Lemma 2.4 is given by \mathbf{C} , one easily sees that q^+ has the following computation rules:

$$\begin{aligned} q^+(\bar{1}u) &:= 00u \\ q^+(0u) &:= 01u \\ q^+(1u) &:= 10u \end{aligned}$$

Analogously, the extracted term q^- of the statement $\forall_x {}^{\text{co}}\mathbf{I}x \rightarrow {}^{\text{co}}\mathbf{I}(\frac{x}{2} - \frac{1}{4})$ is characterised by

$$\begin{aligned} q^-(\bar{1}u) &:= \bar{1}0u \\ q^-(0u) &:= 0\bar{1}u \\ q^-(1u) &:= 00u. \end{aligned}$$

In the direct proof of `sdlim` below we will make use of the following case-distinction. To shorten the extracted term we outsource this case-distinction into a separate lemma.

Lemma 3.6 (TripleCases). *For $x \in {}^{\text{co}}\mathbf{I}$*

$$x \in \left[\frac{1}{8}, 1\right] \vee x \in \left[-1, -\frac{1}{8}\right] \vee x \in \left[-\frac{1}{4}, \frac{1}{4}\right]$$

Proof. Triple application of ${}^{\text{co}}\mathbf{I}^-$ to $x \in {}^{\text{co}}\mathbf{I}$ gives $d_1, d_2, d_3 \in \mathbf{Sd}$ and $y' \in {}^{\text{co}}\mathbf{I}$ such that $x = \frac{4d_1 + 2d_2 + d_3 + y'}{8}$. The claim follows by case-distinction on d_1, d_2 and d_3 . Namely writing $x = d_1 d_2 d_3 y'$ we have

$$\left. \begin{array}{l} 11d_3y' \\ 10d_3y' \\ 1\bar{1}1y' \\ 1\bar{1}0y' \\ 011y' \\ 010y' \end{array} \right\} \rightarrow \frac{1}{8} \leq x \quad \left. \begin{array}{l} \bar{1}\bar{1}d_3y' \\ \bar{1}0d_3y' \\ \bar{1}\bar{1}1y' \\ \bar{1}\bar{1}0y' \\ 0\bar{1}\bar{1}y' \\ 0\bar{1}0y' \end{array} \right\} \rightarrow x \leq -\frac{1}{8} \quad \left. \begin{array}{l} 00d_3y' \\ \bar{1}\bar{1}1y' \\ \bar{1}\bar{1}\bar{1}y' \\ 01\bar{1}y' \\ 0\bar{1}\bar{1}y' \end{array} \right\} \rightarrow -\frac{1}{4} \leq x \leq \frac{1}{4}.$$

□

We omit the extracted term in Haskell here since it is quite long and unreadable due to the 17 case-distinctions. The computational content is basically the diagram in the proof above. With these preparations we are now able to give the direct proof of Theorem 3.2.

SdLim, direct. We show that

$$\forall_x \left(\exists_{xs; M \in \mathbf{Mon}} \left(\forall_{n \in \mathbf{T}} (xs \ n) \in {}^{\text{co}}\mathbf{I} \wedge \forall_{p \in \mathbf{T}} \forall_{n \geq Mp} |x - (xs \ n)| \leq 2^{-p} \right) \rightarrow {}^{\text{co}}\mathbf{I}x \right),$$

which is equivalent. We use (\star) with P the premise of the formula above:

$$Px := \exists_{xs; M \in \mathbf{Mon}} \left(\forall_{n \in \mathbf{T}} (xs \ n) \in {}^{\text{co}}\mathbf{I} \wedge \forall_{n \geq M(p)} |x - (xs \ n)| \leq 2^{-p} \right)$$

Again, we need to prove the second premise, namely

$$\forall_x \left(Px \rightarrow \exists_{d \in \mathbf{Sd}, x'} \left(x' \in ({}^{\text{co}}\mathbf{I} \cup P) \wedge |x'| \leq 1 \wedge x = \frac{d + x'}{2} \right) \right)$$

So let x, xs and $M \in \mathbf{Mon}$ be given and assume $\forall_{n \in \mathbf{T}}(xs\ n) \in {}^{\text{co}}\mathbf{I}$ and $\forall_{p \in \mathbf{T}} \forall_{n \geq M(p)} |(xs\ n) - x| \leq 2^{-p}$. We use the lemma above with $(xs(M(4))) \in {}^{\text{co}}\mathbf{I}$ and get three cases: (i) $\frac{1}{8} \leq xs(M(4))$. In this case we choose

$$d := 1 \text{ and } x' := 2x - 1.$$

Then $|x'| \leq 1$ and $x = \frac{d+x'}{2}$ follow directly. We show that Px' , so we define

$$ys\ n := 2(xs((M(4)) \sqcup n)) - 1,$$

where $m \sqcup l := \max\{m, l\}$ and $N(p) := M(p+1) \in \mathbf{Mon}$. The statement $\forall_{n \geq Np} |ys(n) - x'| \leq 2^{-p}$ is a direct consequence of $\forall_{n \geq M(p)} |xs(n) - x| \leq 2^{-p}$ and it remains to show $\forall_{n \in \mathbf{T}}(ys\ n) \in {}^{\text{co}}\mathbf{I}$. We calculate

$$ys\ n = 4 \left(\frac{xs(M(4) \sqcup n)}{2} - \frac{1}{4} \right),$$

and conclude $\frac{1}{4}(ys\ n) \in {}^{\text{co}}\mathbf{I}$ by Lemma 3.5. Furthermore, we have $xs(M(4)) \geq \frac{1}{8}$ and $\forall_{n \geq M(4)} |xs(n) - x| \leq \frac{1}{16}$ and therefore

$$\begin{aligned} xs(M(4) \sqcup n) &= (xs(M(4) \sqcup n) - x) + (x - xs(M(4))) + xs(M(4)) \\ &\geq -\frac{1}{16} - \frac{1}{16} + \frac{1}{8} = 0. \end{aligned}$$

Hence $0 \leq xs(M(4) \sqcup n) \leq 1$, which implies $|\frac{xs(M(4) \sqcup n)}{2} - \frac{1}{4}| \leq \frac{1}{4}$ and by double application of Lemma 3.4 we finally get $ys\ n \in {}^{\text{co}}\mathbf{I}$.

(ii) $xs(M(4)) \leq -\frac{1}{8}$. In this case we define

$$d := -1, \quad x' := 2x + 1, \quad ys\ n := (2xs(M(4) \sqcup n) + 1), \quad Np := M(p+1).$$

The proof in this case is analogous to the proof of the first case.

(iii) $-\frac{1}{4} \leq xs(M(4)) \leq \frac{1}{4}$. We define

$$d := 0 \quad \text{and} \quad x' := 2x.$$

Again we show Px' , namely

$$\exists_{ys; N \in \mathbf{Mon}} (\forall_{n \in \mathbf{T}}(ys\ n) \in {}^{\text{co}}\mathbf{I} \wedge \forall_{n \geq Np} |x' - (ys\ n)| \leq 2^{-p}).$$

To this end we define

$$ys\ n := 2xs(M(4) \sqcup n) \quad \text{and} \quad Np := M(p+1).$$

Again, the right side of the conjunction follows from the assumptions. For the left side consider

$$|2(ys\ n)| = |xs(M(4) \sqcup n)| \leq |xs(M(4) \sqcup n) - x| + |x - xs(M(4))| + |xs(M(4))| \leq \frac{1}{2},$$

which implies $(ys\ n) \in {}^{\text{co}}\mathbf{I}$ by Lemma 3.4. \square

The extracted term is

```

coilim :: (((Pos -> Nat), (Nat -> Str)) -> Str)
coilim (m,us0) = aiCoRec (m,us0)
  (\ mus1 -> (case mus1 of
    { (,) m us -> (case (cTripleCases (us (m 4))) of
      { Left() -> (cSdLimCaseR m us) ;
        Right(Left ()) ->(cSdLimCaseL m us) ;
        Right(Right()) -> (cSdLimCaseM m us)}}).

```

The terms `cSdLimCaseR`, `cSdLimCaseL` and `cSdLimCaseM` are given by

```

cSdLimCaseR :: ((Pos -> Nat) -> ((Nat -> Str) ->
  (Sd, (Either Str ((Pos -> Nat), (Nat -> Str))))))
cSdLimCaseR m0 us1 = (SdR ,
  (Right ((\ p2 -> (m0 (p2 + 1))) ,
    (\ n2 -> (cCoIToCoIDoublePlusOne (us1 ((m0 3) + n2)))))))

cSdLimCaseM :: ((Pos -> Nat) -> ((Nat -> Str) ->
  (Sd, (Either Str ((Pos -> Nat), (Nat -> Str))))))
cSdLimCaseM m0 us1 = (SdM ,
  (Right ((\ p2 -> (m0 (p2 + 1))) ,
    (\ n2 -> (cCoIToCoIDouble (us1 ((m0 3) + n2)))))))

cSdLimCaseL :: ((Pos -> Nat) -> ((Nat -> Str) ->
  (Sd, (Either Str ((Pos -> Nat), (Nat -> Ai))))))
cSdLimCaseL m0 us1 = (SdL ,
  (Right ((\ p2 -> (m0 (p2 + 1))) ,
    (\ n2 -> (cCoIToCoIDoubleMinusOne (us1 ((m0 3) + n2)))))))

```

In the following we will discuss the computational content, we will denote it by `Lim`, in more detail. It has the type

$$\text{Lim} : (\mathbb{Z}^+ \rightarrow \mathbb{N}) \rightarrow (\mathbb{N} \rightarrow \mathbf{Str}) \rightarrow \mathbf{Str}.$$

It takes as inputs the modulus of convergence and the sequence of streams and returns the stream representing the limit. In order to give a more readable characterisation of `Lim`, we define the following sets

$$\begin{aligned}
\mathbf{R} &:= \{11v, 10v, 1\bar{1}1v, 1\bar{1}0v, 011v, 010v \mid v : \mathbf{Str}\} \\
\mathbf{M} &:= \{00v, \bar{1}11v, 1\bar{1}\bar{1}v, 01\bar{1}v, 0\bar{1}1v \mid v : \mathbf{Str}\} \\
\mathbf{L} &:= \{\bar{1}\bar{1}v, \bar{1}0v, \bar{1}1\bar{1}v, \bar{1}10v, 0\bar{1}\bar{1}v, 0\bar{1}0v \mid v : \mathbf{Str}\}.
\end{aligned}$$

which correspond to the intervals from Lemma 3.6. According to the proof we then have the following rule for `Lim`:

$$\text{Lim } M \ F := \begin{cases} C \ 1 \ (\text{Lim } \lambda_p M(p+1) \ \lambda_n (\text{DD} q^- F(M(4) \sqcup n))) & \text{if } F(M(4)) \in \mathbf{R} \\ C \ 0 \ (\text{Lim } \lambda_p M(p+1) \ \lambda_n (\text{DF}(M(4) \sqcup n))) & \text{if } F(M(4)) \in \mathbf{M} \\ C \ \bar{1} \ (\text{Lim } \lambda_p M(p+1) \ \lambda_n (\text{DD} q^+ F(M(4) \sqcup n))) & \text{if } F(M(4)) \in \mathbf{L} \end{cases}$$

The functions `D`, q^+ and q^- are the computational content of the lemmas above. Note that the definition of the new sequence is not unique. For reasons of efficiency one should be flexible with the choice of the new sequence, which is called ys in the proof above. For example by choosing ys one can replace $M(4) \sqcup n$ by $M(4) + n$. The efficiency depends on the concrete sequence. In the Minlog file we have chosen $M(4) + n$ because the proofs are simpler with the addition instead of the maximum.

3.2 Indirect approach

Now we redo the proof using translations between the Cauchy and sd-representation. First we state the completeness of Cauchy-reals, which will be used.

Theorem 3.7 (RealComplete). *Assume xs and $M \in \mathbf{Mon}$ such that $\forall_{n \in \mathbf{T}} (xs\ n) \in \mathbf{R}$ and $\mathbf{Cauchy}(xs, M)$ then there exists $x \in \mathbf{R}$ such that $\mathbf{Conv}(xs, x, M)$.*

Proof. We refer to Theorem 2.3 in [Sch03]. □

The extracted term is:

```
cRealComplete :: ((Nat -> Rea) -> ((Pos -> Nat) -> Rea))
cRealComplete xs0 m1 = RealConstr
  (\ n -> (realSeq (xs0 n)
    (realMod (xs0 n) (cNatPos n))))
  (\ p -> ((m1 (p + 1)) 'max' ((p + 1) + 1)))
```

Note the following: Given witnesses $(as_n, M_n)_n$ to $\forall_{n \in \mathbf{T}} (xs\ n) \in \mathbf{R}$, the rational sequence witnessing the limit is given by $as\ n := as_n(M_n\ n)$ and the Cauchy-modulus of the limit-real is given by $Np := \max(M(p+1), p+2)$ where M is the modulus of convergence. As preparation for the indirect proof we state some elementary properties of limits and sequences that we will need but do not have any computational content.

Lemma 3.8. *Assume xs is a sequence of reals, x is another real and $M \in \mathbf{Mon}$ such that $\mathbf{Conv}(xs, x, M)$. Then we have*

(i) $\mathbf{Cauchy}(xs, N)$, where $Np := M(p+1)$,

(ii) $\forall_{n \in \mathbf{T}} |(xs\ n)| \leq 1 \rightarrow |x| \leq 1$ and

(iii) $\forall_{y, N \in \mathbf{Mon}} (\mathbf{Conv}(xs, y, N) \rightarrow x = y)$.

Proof. (i) Follows directly by using the triangle inequality and the definitions. (ii) For arbitrary $p \in \mathbf{T}$ let $n := M(p)$. Then $|x| \leq |x - xs(n)| + |xs(n)| \leq 2^{-p} + 1$. As $p \in \mathbf{T}$ is arbitrary it follows $|x| \leq 1$. (iii) We have $xs(n) - x$ converges to 0 with modulus M and $xs(n) - y$ converges to 0 with modulus N . Therefore, $x - y$ converges to zero with modulus $p \mapsto \max\{M(p+1), N(p+1)\}$. I.e. $|x - y| \leq 2^p$ for all p and therefore $x - y = 0$. □

Using the lemmas above, the indirect proof of the convergence theorem becomes quite short:

SdLim, indirect. Assume $x, xs, M \in \mathbf{Mon}, \forall_{n \in \mathbf{T}} (xs\ n) \in {}^{\text{co}}\mathbf{I}$ and $\forall_{n \geq M(p)} |(xs\ n) - x| \leq 2^{-p}$. We apply Theorem 2.7 to $\forall_{n \in \mathbf{T}} (xs\ n)$ and get

$$\forall_{n \in \mathbf{T}} xs \in \mathbf{R} \wedge |xs\ n| \leq 1.$$

By the lemma above xs is a Cauchy-sequence. So we apply Theorem 3.7 to get $y \in \mathbf{R}$ with $\mathbf{Conv}(xs, y, N)$. By the above lemma $x = y \in \mathbf{R}$ and $|y| \leq 1$, so $x \in {}^{\text{co}}\mathbf{I}$ by Theorem 2.10 and 2.3. □

The extracted term for this proof is given by

```
cCoILim :: ((Pos -> Nat) -> ((Nat -> Str) -> Str))
cCoILim m g = cCsToStr (cRealComplete
  (\ n -> (cStrToCs (g n)))
  (\ p -> (m (p + 1)))),
```

i.e. given a sequence $u_0 u_1 \dots$ of **Sd**-streams we apply **cRealComplete** to the sequence of translated stream $\mathbf{cStrToCs}(u_0) \mathbf{cStrToCs}(u_1) \dots$ and then translate the result back.

3.3 Comparison

We now compare the two algorithms obtained by the direct and indirect method. To understand the results of the runtime-experiments we analyze the *lookahead* of the algorithms first. Both limit algorithms have a sequence $F : \mathbb{N} \rightarrow \mathbf{Str}$ of streams and a modulus $M : \mathbb{Z}^+ \rightarrow \mathbb{N}$ of convergence as inputs and they produce one output-stream. Here, the lookahead for some $n \in \mathbb{N}$ is given by two natural numbers $m_0, m_1 \in \mathbb{N}$. Namely, to compute the first n output digits we need the first m_0 digits of the first m_1 elements of F . Unfolding `cRealComplete` in the definition of the indirect case leads to

$$\text{cCoILim}(M, F) = \text{cCsToStr} \left(\left(\sum_{i=1}^n \frac{(F(n))_i}{2^i} \right)_n, \lambda_p M(p+2) \right),$$

where `cCsToStr`(as, M) compares $as(M/3)$ with ± 0.25 . Hence, to compute the n -th digit of `cCoILim`(M, F) we need to examine the first $M(n+4)$ digits of $F(M(n+4))$. The algorithm in the direct case was given by

$$\text{Lim } M F := \begin{cases} C \ 1 \ (\text{Lim } \lambda_p M(p+1) \ \lambda_n (\text{DD}q^- F(M(4) \sqcup n))) & \text{if } F(M(4)) \in \mathbf{R} \\ C \ 0 \ (\text{Lim } \lambda_p M(p+1) \ \lambda_n (\text{DF}(M(4) \sqcup n))) & \text{if } F(M(4)) \in \mathbf{M} \\ C \ \bar{1} \ (\text{Lim } \lambda_p M(p+1) \ \lambda_n (\text{DD}q^+ F(M(4) \sqcup n))) & \text{if } F(M(4)) \in \mathbf{L} \end{cases}$$

By examining the defining equations of \mathbf{D} , q^\pm one easily sees that all these functions needs at most the first $n+1$ digits of the input stream to compute the first n digits of the output stream. For the first digit we need to decide whether $F(M(4))$ is in \mathbf{R}, \mathbf{M} or \mathbf{L} which requires the first three digits of $F(M(4))$, since we apply `cCoITripleClosure`. All in all, to compute the n -th digit of `Lim` $M F$ we need to examine the first $3n$ digits of $F(M(n+3))$. This follows as M is monotone and hence $M(4) \sqcup \dots \sqcup M(n+3) = M(n+3)$.

As we can see, in the direct case the lookahead depends in a way linearly on the modulus M of convergence, whereas in the indirect case it depends quadratically on M . Furthermore, if the modulus of convergence is asymptotically lower than $\lambda_n 3n$, the indirect algorithm should outperform the direct one.

As a first test we run both algorithms in Haskell on the constant sequence $F := \lambda_n u_0$ which converges with the constant modulus $\lambda_p 0$. Here u_0 is a pseudo-random stream of \mathbf{Sd} generated with the Haskell `System.Random` package. To test the dependence of the two algorithms on the modulus of convergence we artificially set different moduli and compute different amounts of digits. All measurements are the average for $n = 10$ tries with different random numbers in seconds.

Mod	50 digits	100 digits	200 digits
$\lambda_p p$	1.78	12.2	87
$\lambda_p p^2$	1.84	12.7	90
$\lambda_p p^3$	2.25	13.4	95

Figure 1: First test - Constant sequences with different moduli for the direct algorithm

As a second experiment we take the geometric series $(x^n)_n$ for some $|x| \leq 0.5$. This is a Cauchy-sequence converging to 0 with modulus $\lambda_p p$, since for $n \leq m$ and $|x| \leq 0.5$ we have

$$|x^n - x^m| \leq |x^n| |1 - x^{n-m}| \leq \frac{1}{2^n}.$$

Again we generate pseudorandom sequences u and here we put a 0 in front to ensure that the absolute value is bounded by 0.5. Then we run both algorithms for the sequence F

Mod	10 digits	20 digits	50 digits	100 digits
$\lambda_p p$	-	< 0.05	0.075	0.21
$\lambda_p p^2$	0.084	0.41	16.38	1140
$\lambda_p p^3$	4.3	503	>1500	-

Figure 2: Constant sequences with different moduli for the indirect algorithm

given by

$$F\ 0 := 0 :: u \quad F(n+1) = \text{cCoIMult}(0 :: u)(F\ n),$$

where cCoIMult is the algorithm from [Sch22]. The results below are the average over $n = 15$ tests. The direct algorithm did not terminate in a reasonable amount of time (≤ 30 minutes) for $n \geq 30$ digits. As expected the indirect algorithm is better here, since the modulus of convergence is the identity here.

digits	indirect	direct
5	0.74	0.69
10	3.3	23.4
20	26	1227
30	87	>1500
40	239	-
50	502	-

Figure 3: Second test - Geometric sequence

4 Applications

4.1 Heron's method

To show an application of the two algorithms extracted in the last section, we define the Heron sequence and show that it converges to the square root.

Definition 4.1 (Heron). *We define $H : \mathbb{R} \rightarrow \mathbb{N} \rightarrow \mathbb{R}$ by the computation rules*

$$H(x, 0) := 1, \quad H(x, n+1) := \frac{1}{2} \left(H(x, n) + \frac{x}{H(x, n)} \right).$$

For every non-negative x the sequence $H(x, \cdot) =: H(x) : \mathbb{N} \rightarrow \mathbb{R}$ is the sequence, we get from Heron's method with initial value 1. Note that H is well-defined for non-negative x since $H(x, n) \geq 2^{-n}$.

Lemma 4.2. *For every $x \in [0, 1]$, $H(x)$ converges to \sqrt{x} with modulus $\iota : \mathbb{Z}^+ \rightarrow \mathbb{N}$. Furthermore we have that*

$$\forall_{n \in \mathbf{T}} \sqrt{x} \leq H(x, n).$$

Proof. Let $x \in [0, 1]$ be given. We define $\Delta(x, n) := H(x, n) - \sqrt{x}$. We calculate

$$\begin{aligned} \Delta(x, n+1) &= \frac{1}{2} \left(H(x, n) + \frac{x}{H(x, n)} \right) - \sqrt{x} = \frac{(H(x, n))^2 - 2H(x, n)\sqrt{x} + x}{2H(x, n)} \\ &= \frac{(\Delta(x, n))^2}{2H(x, n)}. \end{aligned}$$

By induction on n we immediately get $0 \leq H(x, n)$ and therefore $0 \leq \Delta(x, n+1)$. Since $\Delta(x, 0) = 1 - \sqrt{x} \geq 0$ we have $\forall_{n \in \mathbf{T}} \sqrt{x} \leq H(x, n)$. Furthermore, we calculate:

$$\begin{aligned} \Delta(x, n+1) &= \frac{(\Delta(x, n))^2}{2H(x, n)} = \frac{1}{2} \Delta(x, n) \frac{\Delta(x, n)}{H(x, n)} = \frac{1}{2} \Delta(x, n) \left(1 - \frac{\sqrt{x}}{H(x, n)}\right) \\ &\leq \frac{1}{2} \Delta(x, n) \end{aligned}$$

Therefore, by induction we have $|H(x, n) - \sqrt{x}| = \Delta(x, n) \leq 2^{-n}$ and this implies

$$\forall_{p \in \mathbf{T}} \forall_{n \geq p} |H(x, n) - \sqrt{x}| \leq 2^{-p},$$

i.e. $H(x)$ converges to \sqrt{x} with modulus ι . □

This lemma by itself does not have any computational content, but it states that ι is a modulus of convergence of Hx to \sqrt{x} . In some special cases we can improve on the modulus.

Definition 4.3 (Poslog). *For a positive integer p we define $\mathbf{poslog}(p)$ as the least natural number n with $p \leq 2^n$. Equivalently it is the number of digits in the binary representation.*

One possibility to implement the function \mathbf{poslog} is to define an auxiliary function $\mathbf{auxlog} : \mathbb{Z}^+ \rightarrow \mathbb{N} \rightarrow \mathbb{N}$ with the computation rules

$$\mathbf{auxlog} \ p \ n := \begin{cases} n, & \text{if } p \leq 2^n \\ \mathbf{auxlog} \ p \ (n+1), & \text{otherwise,} \end{cases}$$

and then set $\mathbf{poslog}(p) := \mathbf{auxlog} \ p \ 0$.

Proposition 4.4. *If $x \in [\frac{1}{4}, 1]$ then $\mathbf{poslog} : \mathbb{Z}^+ \rightarrow \mathbb{N}$ is a modulus of convergence of $H(x)$ to \sqrt{x} .*

Proof. Let $x \in [\frac{1}{4}, 1]$. From Lemma 4.2 we know $\forall_{n \in \mathbf{T}} \sqrt{x} \leq H(x, n)$ and therefore $\forall_{n \in \mathbf{T}} \frac{1}{2} \leq H(x, n)$. In the proof of Lemma 4.2 the formula

$$\Delta(x, n+1) = \frac{(\Delta(x, n))^2}{2H(x, n)}$$

is proven. This implies $\Delta(x, n+1) \leq (\Delta(x, n))^2$. Since $\frac{1}{2} \leq \sqrt{x}$, by induction we get

$$\Delta(x, n) \leq 2^{-2^n}$$

for all $n \in \mathbf{T}$. Hence for given p and $n \geq \mathbf{poslog}(p)$ we get $p \leq 2^n$ and

$$|H(x, n) - \sqrt{x}| = \Delta(x, n) \leq 2^{-2^n} \leq 2^{-p}.$$

□

Lemma 4.5. *For all $x \in {}^{co}\mathbf{I}$ with $\frac{1}{16} \leq x$ we have $\forall_{n \in \mathbf{T}} H(x, n) \in {}^{co}\mathbf{I}$. Expressed as a formula*

$$\forall_{x \in {}^{co}\mathbf{I}} \left(\frac{1}{16} \leq x \rightarrow \forall_{n \in \mathbf{T}} H(x, n) \in {}^{co}\mathbf{I} \right).$$

Proof. We use the results of [MS15], [SW21] and of Section 3.3 from [Wie17]. Namely we have

$$\forall_{x \in {}^{\text{co}}\mathbf{I}, y \in {}^{\text{co}}\mathbf{I}} \frac{x+y}{2} \in {}^{\text{co}}\mathbf{I}. \quad (4.1)$$

In Minlog this theorem is implemented in `average.scm` in the folder `examples/analysis` and has the name `CoIAverage`. Furthermore

$$\forall_{x \in {}^{\text{co}}\mathbf{I}, y \in {}^{\text{co}}\mathbf{I}} \left(|x| \leq y \rightarrow \frac{1}{4} \leq y \rightarrow \frac{x}{y} \in {}^{\text{co}}\mathbf{I} \right) \quad (4.2)$$

is proven there. In Minlog this theorem is implemented in `sddiv.scm` in the folder `examples/analysis` and has the name `CoIDiv`. Using these, the proof of this lemma is done by induction on n : For $n = 0$ it is easy since $H(x, 0) = 1$ and $1 \in {}^{\text{co}}\mathbf{I}$. For any total n we have

$$H(x, n+1) = \frac{1}{2} \left(H(x, n) + \frac{x}{H(x, n)} \right).$$

By Lemma 4.2 we get $\sqrt{x} \leq H(x, n)$ and therefore

$$\sqrt{\frac{1}{16}} = \frac{1}{4} \leq H(x, n) \quad \text{and} \quad x \leq \sqrt{x} \leq H(x, n).$$

Additionally, by the induction hypothesis, $H(x, n) \in {}^{\text{co}}\mathbf{I}$. By (4.2) we have $\frac{x}{H(x, n)} \in {}^{\text{co}}\mathbf{I}$, so with (4.1) we get $H(x, n+1) \in {}^{\text{co}}\mathbf{I}$. \square

By `cCoIAverage` and `cCoIDiv` we denote the computational content of (4.1) and (4.2). Each of these terms takes two streams of reals and returns a stream of their average and their quotient, respectively. Then the extracted term is

```
cCoIHeron :: (Str -> (Nat -> Str))
cCoIHeron u0 n1 = natRec n1
  cCoIOne
  (\ n2 -> (\ u3 -> (cCoIAverage u3 (cCoIDiv u0 u3))))
```

Informally the computational content `Heron` is defined by recursion:

$$\begin{aligned} \text{Heron } v \ 0 &:= [1, 1, \dots] \\ \text{Heron } v \ (n+1) &:= \text{cCoIAverage}(\text{Heron } v \ n)(\text{cCoIDiv } v \ (\text{Heron } v \ n)) \end{aligned}$$

Which is Definition 4.1 in the notation of streams.

Theorem 4.6.

$$\forall_{x \in {}^{\text{co}}\mathbf{I}} (0 \leq x \rightarrow \sqrt{x} \in {}^{\text{co}}\mathbf{I})$$

Proof. We apply (\star) with

$$Px := \exists_{y \in {}^{\text{co}}\mathbf{I}} (0 \leq y \wedge \sqrt{y} = x).$$

To show the goal formula, we need to show the second premise, namely for all x

$$\exists_{y \in {}^{\text{co}}\mathbf{I}} (0 \leq y \wedge \sqrt{y} = x) \rightarrow \exists_{d \in \mathbf{Sd}, x'} \left(x' \in ({}^{\text{co}}\mathbf{I} \cup P) \wedge |x'| \leq 1 \wedge x = \frac{d+x'}{2} \right).$$

Let $y \in {}^{\text{co}}\mathbf{I}$ with $0 \leq x$ and $\sqrt{y} = x$ be given. Triple application of ${}^{\text{co}}\mathbf{I}^-$ to $y \in {}^{\text{co}}\mathbf{I}$ yields $d_1, d_2, d_3 \in \mathbf{Sd}$ and $y' \in {}^{\text{co}}\mathbf{I}$ with $y = d_1 d_2 d_3 y'$. We distinguish three different cases: If y has one of the forms $\bar{1}d_2 d_3 y', 0\bar{1}d_3 y'$ or $00\bar{1}y'$ we have $y \leq 0$ and therefore $x = \sqrt{y} = 0$.

Hence we define $d := 0$ and $x' := 0$ and the claim follows immediately. If y has one of the forms $000y'$, $001y'$, $01\bar{1}y'$ or $1\bar{1}\bar{1}y'$ we can rewrite $y = 00ey'$ for some $e \in \{0, 1\}$. Here we define $d := 0$ and $x' := \sqrt{\frac{e+y'}{2}}$. Then

$$x = \sqrt{y} = \sqrt{\frac{e+y'}{8}} = \frac{\sqrt{\frac{e+y'}{2}}}{2} = \frac{d+x'}{2}.$$

Furthermore $\frac{e+y'}{2} \in {}^{\text{co}}\mathbf{I}$ by Lemma 2.4 and $0 \leq \frac{e+y'}{2}$ since $0 \leq y = \frac{e+y'}{8}$. Altogether we get Px' . The remaining case is that y has one of the forms $010y'$, $011y'$, $1\bar{1}1y'$, $1\bar{1}0y'$, $10d_3y'$ or $11d_3y'$. In that case we have $\frac{1}{8} \leq y$. Hence by Lemma 4.5 $\forall_{n \in \mathbf{T}} {}^{\text{co}}\mathbf{I}(H(y, n))$ and we know that $H(y)$ converges to \sqrt{y} with modulus $\iota : \mathbb{Z}^+ \rightarrow \mathbb{N}$ by Lemma 4.2. Thus we use Theorem 3.2 to get $x \in {}^{\text{co}}\mathbf{I}$ and by one application of ${}^{\text{co}}\mathbf{I}^-$

$$\exists_{d \in \mathbf{Sd}, x' \in {}^{\text{co}}\mathbf{I}} \left(|x'| \leq 1 \wedge x = \frac{d+x'}{2} \right),$$

which proves the goal formula. \square

We omit the description of the extracted term as Haskell code as it is quite long due to the case distinctions. But we give an informal description of the extracted term: By the definitions of **cSdLim** as the computational content of Theorem 3.2 and **Heron** as the computational content of Lemma 4.5 we have the following rules for the computational content **sqrt** : **Str** \rightarrow **Str** of this theorem:

$$\begin{aligned} \text{sqrt}(\bar{1}u) &:= [0, 0, \dots] \\ \text{sqrt}(0\bar{1}u) &:= [0, 0, \dots] \\ \text{sqrt}(00u) &:= 0 \text{ sqrt } u \\ \text{sqrt}(01\bar{1}u) &:= 0 \text{ sqrt } 1u \\ \text{sqrt}(1\bar{1}\bar{1}u) &:= 0 \text{ sqrt } 1u \\ \text{sqrt } u &:= \text{cSdLim } \iota (\text{Heron } u) \end{aligned}$$

The last rule shall only be applied if the other rules do not fit.

4.2 Multiplication

Our last application is motivated by Helmut Schwichtenberg and the Minlog file **sdmult.scm** in **examples/analysis**. There it is proven that for any $x, y \in {}^{\text{co}}\mathbf{I}$ the product xy is also in ${}^{\text{co}}\mathbf{I}$. In the following we use the limit-theorem to formulate another proof of this theorem. Our approach is based on repeated applications of ${}^{\text{co}}\mathbf{I}^-$ to $y \in {}^{\text{co}}\mathbf{I}$, namely

$$xy = \frac{xd_1 + xy_1}{2} = \frac{x(d_1 + \frac{d_1}{2}) + x\frac{y_2}{2}}{2} = \dots = x \sum_{i=1}^n \frac{d_i}{2^i} + x\frac{y_n}{2^n},$$

and the sequence $xs \ n := \sum_{i=1}^n \frac{d_i}{2^i}$ converges to y . In order to realize this idea we first define a constant **Sum** : $\mathbb{L}(\mathbf{Sd}) \rightarrow \mathbb{Q}$ by

$$\mathbf{Sum}[] = 0 \quad \mathbf{Sum}l := \sum_{i=1}^{\text{lth}(l)} \frac{(l)_i}{2^i},$$

where $\text{lth} : \mathbb{L} \rightarrow \mathbb{N}$ is the length-function for lists and $(l)_i$ is the i -th element of the list l . We prove the following.

Lemma 4.7 (CoIMultSum). *Let $x \in {}^{\text{co}}\mathbf{I}$ then for all $l : \mathbb{L}(\mathbf{Sd})$ in \mathbf{T} we have*

$$x \cdot \mathbf{Sum}l \in {}^{\text{co}}\mathbf{I}.$$

Proof. We use the theorem **CoIAverage**, which was already mention in the proof of Lemma 4.5 and **CoISdTimes** (i.e. $\forall_{x \in {}^{\text{co}}\mathbf{I}, d \in \mathbf{Sd}} dx \in {}^{\text{co}}\mathbf{I}$). The proof is done by induction on $l \in \mathbf{T}$. Namely if $l = []$ then $x \cdot 0 = 0 \in {}^{\text{co}}\mathbf{I}$ (see **CoIZero** in Minlog). Now assume that $x \cdot \mathbf{Sum}l \in {}^{\text{co}}\mathbf{I}$ and $d \in \mathbf{Sd}$. We calculate

$$x \cdot \mathbf{Sum}(d :: l) = x \sum_{i=1}^{n+1} \frac{(l)_i}{2^i} = \frac{1}{2} \left(x \sum_{i=1}^n \frac{(l)_{i+1}}{2^i} + x \cdot (l)_1 \right).$$

Now we can apply **CoIAverage**, namely $x \cdot l_1 \in {}^{\text{co}}\mathbf{I}$ by **CoISdTimes** and the first summand is in ${}^{\text{co}}\mathbf{I}$ by the induction hypothesis. \square

Note that the induction hypothesis is applied to the list $(l)_2 :: \dots :: (l)_{n+1}$, so the stream computed in the previous step is not used. Therefore, the runtime of the algorithm must be at least quadratic in the length of the list in the output. We will later see that the runtime of the algorithm that is obtained is worse than the one extracted in [Sch22]. The Haskell-translation of the extracted term is

```
cCoIMultSum :: Str -> [Sd] -> Str
cCoIMultSum u0 l1 = listRec
  l1
  cCoIZero
  (\ s2 -> (\ l3 -> (\ u4 -> (cCoIAverage (cCoISdTimes s2 u0) u4)
    )))
```

And the computational content of the lemma, here denoted $\mathbf{f} : \mathbf{Str} \rightarrow \mathbb{L}(\mathbf{Sd}) \rightarrow \mathbf{Str}$, can also be represented in the more readable form by

$$\begin{aligned} \mathbf{f}(u, []) &= \mathbf{cCoIZero} \\ \mathbf{f}(u, d :: l) &= \mathbf{cCoIAverage}(\mathbf{f}(u, l), \mathbf{cCoISdTimes}(d, u)), \end{aligned}$$

where **cCoIZero** is analogous to **cCoIOne**:

```
cCoIZero :: Str
cCoIZero = (aiCoRec () (\ g -> (SdM , (Right ())))))
```

The next lemma is basically repeated application of ${}^{\text{co}}\mathbf{I}^-$. The proof is very similar to the proof of Theorem 2.7 and so are the extracted terms.

Lemma 4.8 (CoIToConvSeq). *Let $x \in {}^{\text{co}}\mathbf{I}$ then there exists $G : \mathbb{N} \rightarrow \mathbb{L}(\mathbf{Sd})$ in \mathbf{T} such that*

$$\mathbf{Conv}(\lambda_n \mathbf{Sum}G(n), x, \iota).$$

Proof. For $x \in {}^{\text{co}}\mathbf{I}$ we first show that

$$\exists_{G \in \mathbf{T}} \forall_{n \in \mathbf{T}} \exists_{y \in {}^{\text{co}}\mathbf{I}} \left(\text{lth}(G \ n) = n \wedge x = \mathbf{Sum}(G \ n) + \frac{y}{2^n} \right).$$

By application of **CC** it suffices to show

$$\forall_{n \in \mathbf{T}} \exists_{l \in \mathbf{T}} \exists_{y \in {}^{\text{co}}\mathbf{I}} \left(\text{lth}(l) = n \wedge x = \mathbf{Sum}l + \frac{y}{2^n} \right),$$

which is done by induction on $n \in \mathbf{T}$. If $n = 0$ then choose $y := x$ and $l = []$. Now assume we have $l' \in \mathbf{T}$ with $\text{lth}(l') = n$ and $z \in {}^{\text{co}}\mathbf{I}$ with

$$x = \mathbf{Sum}l' + \frac{z}{2^n}.$$

We apply ${}^{\text{co}}\mathbf{I}^-$ to z and get $d \in \mathbf{Sd}, y \in {}^{\text{co}}\mathbf{I}$ with $z = \frac{y+d}{2}$, then $l := l' \star d$ (Note that \star denotes concatenation of lists) will do the trick. So assume we have G with the properties above and $n \in \mathbf{T}$. Then there exists some $y_n \in {}^{\text{co}}\mathbf{I}$ with $x = \mathbf{Sum}(G\ n) + \frac{y}{2^n}$ and

$$|x - \mathbf{Sum}(G\ n)| \leq \frac{|y_n|}{2^n} \leq 2^{-n}.$$

Hence $\mathbf{Sum}(G \cdot)$ converges to x with modulus PosToNat . \square

The extracted term is

```
cCoIToConvSeq :: Str -> Nat -> [Sd]
cCoIToConvSeq u0 n1 = fst
  (natRec n1 ([], u0)
   (\ n2 -> (\ g -> (case g of
     { (,) 1 u1 -> ((1 ++ ((head u1) : [])) , (tail u1)) }))))
```

Let $g : \mathbf{Str} \rightarrow \mathbb{N} \rightarrow \mathbb{L}(\mathbf{Sd})$ denote a simplified iterative version of the computational content of the last lemma. It can be given by the computation rules

$$g(u, 0) = ([], u) \quad g(C\ s\ v, n+1) = g(v, n) \star s,$$

so this is the function that returns the first n elements of the stream. The last lemma we need states that limits are closed under multiplication, namely:

Lemma 4.9. *Assume $\mathbf{Conv}(ys, y, M)$ and $|x| \leq 1$ then $\mathbf{Conv}(\lambda_n(x \cdot (ys\ n)), xy, M)$.*

The proof is elementary. Now we can apply the limit theorem.

Theorem 4.10. *For all $x, y \in {}^{\text{co}}\mathbf{I}$ we have $x \cdot y \in {}^{\text{co}}\mathbf{I}$.*

Proof. We apply Lemma 4.8 to $y \in {}^{\text{co}}\mathbf{I}$ in order to obtain $G \in \mathbf{T}$ with

$$\mathbf{Conv}(\mathbf{Sum}(G, \cdot), y, \text{PosToNat}).$$

By Lemma 4.9 $x \cdot (\mathbf{Sum}(G\ n)) \in {}^{\text{co}}\mathbf{I}$ for all n and it converges to $x \cdot y$ by Lemma 4.9. Hence, we apply Theorem 3.2. \square

The extracted term from Haskell is given by

```
cCoIMultLim :: Str -> Str -> Str
cCoIMultLim u0 u1 = cCoILim
  id
  (\ n2 -> (cCoIMultSum u0 (cCoIToConvSeq u1 n2))).
```

Now let $\mathbf{Mult} : \mathbf{Str} \rightarrow \mathbf{Str} \rightarrow \mathbf{Str}$ denote the computational content in human-readable. Then

$$\mathbf{Mult}(u, v) := \text{coilim}(\text{PosToNat}, \lambda_n f(u, g(v, n))).$$

We compare the algorithms obtained in this way using the indirect and direct limit theorem with the algorithm from [Sch22] that was obtained from a direct proof. To this end we apply all three algorithms to two randomly generated sequences and measure the runtime in Haskell. The result is the average over $n = 10$ tests. Although the algorithm using the direct limit is better here, the algorithm from [Sch22] still performs best. It seems that in order to obtain efficient algorithms, completeness results should only be used if they are really needed.

number of digits	mult(schwicht)	mult(lim,indirect)	mult(lim,direct)
5	0.056	0.51	0.1
10	0.061	15	0.71
15	0.063	432	11.6
30	0.095	-	60.8
100	0.43	-	-

Figure 4: Third test - Runtime of different multiplication algorithms

5 Conclusion and further work

We presented a formal method for extracting verified algorithms for exact real number arithmetic using different representations. All the proofs up to Section 4 have been carried out in the proof assistant Minlog. Furthermore automatic generation of correctness proofs and translation to Haskell was carried out. Even though the proofs from 4 have only been partially carried out in a proof assistant, the program extraction by hand was still a reliable method to get certified algorithms.

Although algorithms extracted via the indirect method by translations tend to have a low lookahead, they do rely on rational arithmetic, so the direct method should outperform the indirect one in most cases. Our aim was to obtain verified algorithms and we do not claim that our programs are the most efficient. Some inefficiency stems from overestimation of bounds in formal proofs. These can usually be removed by careful analysis of the proofs involved.

Since we have proven that the signed digit representation is closed under multiplication, average, division and limits we can now easily prove that a lot of functions are represented by stream-transformers, e.g. trigonometric functions, using their Taylor-expansion directly as was done for Herons algorithm. Another viable approach to limits should be to use the completeness of metric spaces, i.e. prove that signed-digits-reals satisfy the axioms of a metric space and then use a completeness theorem for abstract metric spaces.

Acknowledgment

The first author would like to thank the Istituto Nazionale di Alta Matematica “Francesco Severi” for their scholarship of his PhD study. This project has received funding from the European Unions Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 731143. In addition it was funded by the FWF project P 32080-N31. Both authors would like to thank Helmut Schwichtenberg for proof reading this paper and for his support during the creation of this paper.

References

- [Ber11] Ulrich Berger. From coinductive proofs to exact real arithmetic: theory and applications. *Log. Methods Comput. Sci.*, 7(1), 2011.
- [BH08] Ulrich Berger and Tie Hou. Coinduction for Exact Real Number Computation. *Theory of Computing Systems*, 43(3):394–409, 2008.
- [BS12] Ulrich Berger and Monika Seisenberger. Proofs, programs, processes. *Theory of Computing Systems*, 51(3):313–329, 2012.

- [CG06] Alberto Ciaffaglione and Pietro Di Gianantonio. A certified, corecursive implementation of exact real numbers. *Theoretical Computer Science*, 351:39–51, 2006.
- [Köp18] Nils Köpp. Automatically verified program extraction from proofs with applications to constructive analysis. MSc Thesis, Ludwig Maximilians University Munich, 2018.
- [Miy17] Kenji Miyamoto. The Minlog System. <http://www.mathematik.uni-muenchen.de/~logik/minlog/index.php>, 2017. [Online; accessed 17 June 2022].
- [MS15] Kenji Miyamoto and Helmut Schwichtenberg. Program extraction in exact real arithmetic. *Mathematical Structures in Computer Science*, 25(Special issue 8):1692–1704, 2015.
- [Sch03] Helmut Schwichtenberg. Constructive analysis with witnesses. In *Proc. NATO Advanced Study Institute*, Marktoberdorf, 2003, pages 323–353, 2003.
- [Sch21] Helmut Schwichtenberg. Computational aspects of Bishop’s constructive mathematics. In D. Bridges, H. Ishihara, H. Schwichtenberg, and M. Rathjen, editors, *Handbook of constructive mathematics*. Cambridge University Press, 2021. Submitted.
- [Sch22] Helmut Schwichtenberg. Logic for exact real arithmetic: multiplication. To appear in *Mathematics for Computation (M4C)* (ed. M. Benini, O. Beyersdorff, M. Rathjen, P. Schuster), World Scientific, Singapore, 2022.
- [SW12] Helmut Schwichtenberg and Stanley S. Wainer. *Proofs and Computations*. Perspectives in Logic. Association for Symbolic Logic and Cambridge University Press, 2012.
- [SW21] Helmut Schwichtenberg and Franziskus Wiesnet. Logic for exact real arithmetic. *Logical Methods in Computer Science*, 17:2, 2021.
- [Wie80] Edwin Wiedmer. Computing with infinite objects. *Theoretical Computer Science*, 10:133–155, 1980.
- [Wie17] Franziskus Wiesnet. Konstruktive Analysis mit exakten reellen Zahlen. MSc Thesis, Ludwig Maximilians University Munich, 2017.
- [Wie18] Franziskus Wiesnet. Introduction to Minlog. In Klaus Mainzer, Peter Schuster, and Helmut Schwichtenberg, editors, *Proof and Computation*, pages 233–288. World Scientific, 2018.
- [Wie21] Franziskus Wiesnet. *The computational content of abstract algebra and analysis*. PhD thesis, Ludwig Maximilians University Munich, 2021.