Program Extraction in Constructive Analysis

Helmut Schwichtenberg

Abstract

We sketch a development of constructive analysis in Bishop's style, with special emphasis on low type-level witnesses (using separability of the reals). The goal is to set up things in such a way that realistically executable programs can be extracted from proofs. This is carried out for (1) the Intermediate Value Theorem and (2) the existence of a continuous inverse to a monotonically increasing continuous function. Using the Minlog proof assistant, the proofs leading to the Intermediate Value Theorem are formalized and realizing terms extracted. It turns out that evaluating these terms is a reasonably fast algorithm to compute, say, approximations of $\sqrt{2}$.

1 Introduction

We are interested in *exact real numbers*, as opposed to floating point numbers. The final goal is to develop the basics of real analysis in such a way that from a proof of an existence formula one can extract a program. For instance, from a proof of the intermediate value theorem we want to extract a program that, given an arbitrary error bound 2^{-k}, computes a rational a where the given function is zero up to the error bound.

Why should we be interested in logic in a study of constructive analysis? There are at least two reasons.

- Obviously we need to be aware of the difference of the classical and the constructive existential quantifier, and try to prove the stronger statements involving the latter whenever possible. Then one is forced to give "constructive" proofs, whose algorithmic content can be "seen" and then used as a basis to formulate a program for computing the solution. This was the point of view in Bishop's classic textbook [4], and more explicitly carried through in Andersson's Master's thesis [1] (based on Palmgren's [9]), with Mathematica as the target programming language.

- However, one can go one step further and automatize the step from the (formalized) constructive proof to the corresponding program. This

can be done by means of the so-called realizability interpretation introduced by Kreisel [7]. The desire to have "mathematics as a numerical language" in this sense was clearly expressed by Bishop in his article [5] (with just that title). There are now many implementations of these ideas, for instance Coq, Isabelle/HOL, Agda, Nuprl, Minlog (cf. Wiedijk's "The Seventeen Provers of the World", to appear as Springer LNAI).

What are the requirements on a constructive logic that should guide us in our design?

- It should be as close as possible to the mathematical arguments we want to use. Variables should carry (functional) types, with free algebras (e.g., natural numbers) as base types. Over these, inductive definitions and the corresponding introduction and elimination axioms should be allowed.

- The constants of the language should denote computable functionals in the Scott-Ershov sense, and hence the higher-order quantifiers should range over their (mathematically correct) domain, the partial continuous functionals.

- The language of the logic should be strong (in the sense of being expressive), but the existence axioms used should be weak.

- Type parameters (ML style) should be allowed, but quantification over types should be disallowed in order to keep the theory predicative. Similarly, predicate variables should be allowed as place-holders for formulas (or more precisely, comprehension terms, that is formulas with some variables abstracted). However, in comprehension terms quantification over predicate variables is not allowed, since this would form a glaring impredicativity: we then would define a predicate (by the comprehension term) with reference to the totality of all predicates, to which the one to be defined belongs.

The Minlog proof assistant (`www.minlog-system.de`, under development in Munich), has been designed to meet these demands. Some information on the logical theory it formalizes can be found in [11]. The present paper reports on experiences in formalizing proofs in constructive analysis and extracting programs from them, using the Minlog proof assistant.

Sections 2 and 3 develop some basic machinery of constructive analysis, in a form suitable for formalization. This includes proofs of the Intermediate Value Theorem in 3.4 and of the fact that every continuous function with a uniform lower bound on its slope has a continuous inverse, in 3.5. In section 4 we report on a formalization of the material in sections 2 and 3, which includes program extraction. After providing in 4.1 some general

proof-theoretical background on program extraction and on necessary optimizations, in 4.2 we display the extracted terms for the proofs leading to the Intermediate Value Theorem, and discuss what they intuitively mean. We then report on experiments using these terms to compute approximations of $\sqrt{2}$.

Related work has recently been done in the "Constructive Coq Repository (C-CoRN)" project at Nijmegen (Barendregt, Geuvers, Wiedijk, Cruz-Filipe [6]). It grew out of the FTA project, where Kneser's constructive proof of the fundamental theorem of algebra was to be formalized. However, program extraction from this proof turned out to be problematic, for a number of reasons, among them:

- Strong extensionality was required, in the form $\forall_{x,y}.f(x)\#f(y) \to x\#y$. However, the missing witness in this formula turned out to be harmful for program extraction.
- The **Set**, **Prop** distinction in Coq was found to be insufficient. In his thesis [6], Cruz-Filipe was forced to introduce a certain variant he called "**CProp**".

Compared with the literature, the novel aspect of the present work is the development of elementary constructive analysis in such a way that witnesses have as low a type level as possible. For example, a continuous function on the reals is determined by its values on the rationals, and hence can be represented by a type-one (rather than type-two) object. This clearly is important for the complexity of the extracted programs.

The subject of the present volume is "Logicism, Intuitionism, and Formalism – What has become of them?" This paper can be seen as providing first steps in an attempt to develop "Mathematics as a numerical language", as advocated by Bishop [5]. Clearly this requires a constructive view of mathematics as advocated by Brouwer, but in the liberal sense of Bishop and Troelstra. But of course, any such attempt would be impossible without the pioneering ideas and concepts from all these foundational schools.

2 Real Numbers

2.1 Reals, Equality of Reals

We shall view a real as a Cauchy sequence of rationals with a separately given modulus.

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

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

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

We shall loosely speak of a real $(a_n)_n$ if the Cauchy modulus M is clear from the context or inessential. Every rational a is tacitly understood as the real represented by the constant sequence $a_n = a$ with the constant modulus $M(k) = 0$.

Definition. Two reals $x := ((a_n)_n, M)$, $y := ((b_n)_n, N)$ are called *equivalent* (or *equal*), which is written $x = y$ if the context makes clear what is meant, if

$$|a_{M(k+1)} - b_{N(k+1)}| \leq 2^{-k} \quad \text{for all } k \in \mathbb{N}$$

We want to show that this is an equivalence relation. Reflexivity and symmetry are clear. For transitivity we use the following lemma:

Lemma. *Two reals $x := ((a_n)_n, M)$, $y := ((b_n)_n, N)$ are equal if and only if*

$$\forall_k \exists_q^{\mathsf{cl}} \forall_{n \geq q} \, |a_n - b_n| \leq 2^{-k}.$$

Lemma. *Equality between reals is transitive.*

Proof. Let $(a_n)_n$, $(b_n)_n$, $(c_n)_n$ be the Cauchy sequences for x, y, z. Assume $x = y$, $y = z$ and pick p, q according to the lemma above. Then $|a_n - c_n| \leq |a_n - b_n| + |b_n - c_n| \leq 2^{-k-1} + 2^{-k-1}$ for $n \geq p, q$. □

2.2 The Archimedean Axiom

For every function on the reals we certainly want compatibility with equality. This however is not always the case; here is an important example.

Lemma (RealBound). *For every real $x := ((a_n)_n, M)$ we can find an upper bound 2^{k_x} on the elements of the Cauchy sequence: $|a_n| \leq 2^{k_x}$ for all n.*

Proof. Let k_x be such that $\max\{\, |a_n| \mid n \leq M(0) \,\} + 1 \leq 2^{k_x}$. Hence $|a_n| \leq 2^{k_x}$ for all n. □

Clearly this assignment of k_x to x is not compatible with equality.

2.3 Nonnegative and Positive Reals

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

$$-2^{-k} \leq a_{M(k)} \quad \text{for all } k \in \mathbb{N}.$$

It is *k-positive* (written $x \in_k \mathbb{R}^+$, or $x \in \mathbb{R}^+$ if k is not needed) if

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

We want to show that both properties are compatible with equality. First we prove a useful characterization of nonnegative reals.

Lemma. *A real $x := ((a_n)_n, M)$ is nonnegative if and only if*

$$\forall_k \exists_p \forall_{n \geq p} - 2^{-k} \leq a_n.$$

Lemma. *If $x \in \mathbb{R}^{0+}$ and $x = y$, then also $y \in \mathbb{R}^{0+}$.*

Proof. Let $x := ((a_n)_n, M)$ and $y := ((b_n)_n, N)$. Assume $x \in \mathbb{R}^{0+}$ and $x = y$, and let k be given. Pick p according to the lemma above and q according to the characterization of equality of reals in 2.1 (both for $k+1$). Then for $n \geq p, q$

$$-2^{-k} \leq -2^{-k-1} + a_n \leq (b_n - a_n) + a_n.$$

Hence $y \in \mathbb{R}^{0+}$ by definition. □

We now show compatibility of positivity with equality. Again we need a lemma:

Lemma. *A real $x := ((a_n)_n, M)$ is k-positive if and only if*

$$\exists_{l,p} \forall_{n \geq p} \, 2^{-l} \leq a_n.$$

For $\forall_{n \geq p} \, 2^{-l} \leq a_n$ write $x \in_{l,p} \mathbb{R}^+$ or $0 <_{l,p} x$.

Lemma. *Positivity of reals is compatible with equality.*

Proof. Assume $0 <_{k,p} x$ and $x = y$, so in particular we have a q such that $\forall_{n \geq q} \, |a_n - b_n| \leq 2^{-k-1}$. Then for $\max(p, q) \leq n$

$$2^{-k-1} = -2^{-k-1} + 2^k \leq (b_n - a_n) + a_n = b_n.$$

Hence $0 <_{k+1, \max(p,q)} y$. □

2.4 Arithmetical Functions

Given real numbers $x := ((a_n)_n, M)$ and $y := ((b_n)_n, N)$, we define each z from the list $x + y$, $-x$, $|x|$, $x \cdot y$, and $\frac{1}{x}$ (the latter only provided that $|x| \in_l \mathbb{R}^+$) as represented by the respective sequence (c_n) of rationals with modulus L:

z	c_n	$L(k)$
$x + y$	$a_n + b_n$	$\max\big(M(k+1), N(k+1)\big)$
$-x$	$-a_n$	$M(k)$
$\lvert x \rvert$	$\lvert a_n \rvert$	$M(k)$
$x \cdot y$	$a_n \cdot b_n$	$\max\big(M(k+1+k_{\lvert y \rvert}), N(k+1+k_{\lvert x \rvert})\big)$
$\frac{1}{x}$ for $\lvert x \rvert \in_l \mathbb{R}^+$	$\begin{cases} \frac{1}{a_n} & \text{if } a_n \neq 0 \\ 0 & \text{if } a_n = 0 \end{cases}$	$M(2(l+1)+k)$,

where 2^{k_x} is the upper bound provided by 2.2.

Lemma. *For reals x, y also $x+y$, $-x$, $|x|$, $x \cdot y$ and (provided that $|x| \in_l \mathbb{R}^+$) also $1/x$ are reals.*

Lemma. *The functions $x + y$, $-x$, $|x|$, $x \cdot y$ and (provided that $|x| \in_l \mathbb{R}^+$) also $1/x$ are compatible with equality.*

Lemma. *For reals x, y, z*

$$\begin{aligned} x + (y + z) &= (x + y) + z & x \cdot (y \cdot z) &= (x \cdot y) \cdot z \\ x + 0 &= x & x \cdot 1 &= x \\ x + (-x) &= 0 & 0 < |x| \to x \cdot \frac{1}{x} &= 1 \\ x + y &= y + x & x \cdot y &= y \cdot x \end{aligned}$$

$$x \cdot (y + z) = x \cdot y + x \cdot z$$

Lemma. *For reals x, y from $x \cdot y = 1$ we can infer $0 < |x|$.*

Proof. Pick k such that $|b_n| \leq 2^k$ for all n. Pick q such that $q \leq n$ implies $1/2 \leq a_n \cdot b_n$. Then for $q \leq n$, $1/2 \leq |a_n| \cdot 2^k$, and hence $2^{-k-1} \leq |a_n|$. □

Lemma. *For reals x, y,*

(a) $x, y \in \mathbb{R}^{0+} \to x + y, x \cdot y \in \mathbb{R}^{0+}$,

(b) $x, y \in \mathbb{R}^{+} \to x + y, x \cdot y \in \mathbb{R}^{+}$,

(c) $x \in \mathbb{R}^{0+} \to -x \in \mathbb{R}^{0+} \to x = 0$.

Proof. (a), (b). Routine. (c). Let k be given. Pick p such that $-2^{-k} \leq a_n$ and $-2^{-k} \leq -a_n$ for $n \geq p$. Then $|a_n| \leq 2^{-k}$. □

2.5 Comparison of Reals

We write $x \leq y$ for $y - x \in \mathbb{R}^{0+}$ and $x < y$ for $y - x \in \mathbb{R}^{+}$. Unwinding the definitions yields that $x \leq y$ is to say that for every k, $a_{L(k)} \leq b_{L(k)} + 2^{-k}$ with $L(k) := \max(M(k), N(k))$, or equivalently (using the characterization of reals in 2.3) that for every k there exists p such that $a_n \leq b_n + 2^{-k}$ for all $n \geq p$. Furthermore, $x < y$ is a shorthand for the presence of k with $a_{L(k+1)} + 2^{-k} \leq b_{L(k+1)}$ with L the maximum of M and N, or equivalently (using the characterization of reals in 2.3) for the presence of k, q with $a_n + 2^{-k} \leq b_n$ for all $n \geq q$; we then write $x <_k y$ (or $x <_{k,q} y$) whenever we want to call these witnesses.

Lemma (RealApprox). $\forall_{x,k} \exists_a |a - x| \leq 2^{-k}$.

Proof. Let $x = ((a_n), M)$. Given k, pick $a_{M(k)}$. We show $|a_{M(k)} - x| \leq 2^{-k}$, that is $|a_{M(k)} - a_{M(l)}| \leq 2^{-k} + 2^{-l}$ for every l. But this follows from $|a_{M(k)} - a_{M(l)}| \leq |a_{M(k)} - a_{M(k+l)}| + |a_{M(k+l)} - a_{M(l)}| \leq 2^{-k} + 2^{-l}$. □

Lemma. $0 \le x$ *and* $0 <_k y$ *imply* $0 <_{k+1} x + y$.

Proof. From $0 \le x$ we have $\forall_l \exists_p \forall_{n \ge p} \; -2^{-l} \le a_n$. From $0 <_k y$ we have some q such that $\forall_{n \ge q} 2^{-k} \le b_n$. Pick p for $k+1$. Then $p, q \le n$ implies $0 \le a_n + 2^{-k-1}$ and $2^{-k-1} \le b_n - 2^{-k-1}$, hence $2^{-k-1} \le a_n + b_n$. □

Lemma. *For reals* x, y, z,

$$\begin{array}{ll} x \le x & x \not< x \\ x \le y \to y \le x \to x = y & x < y \to y < z \to x < z \\ x \le y \to y \le z \to x \le z & x < y \to x + z < y + z \\ x \le y \to x + z \le y + z & x < y \to 0 < z \to x \cdot z < y \cdot z \\ x \le y \to 0 \le z \to x \cdot z \le y \cdot z & \end{array}$$

Proof. From 2.4. □

Lemma. $x \le y \to y <_k z \to x <_{k+1} z$.

Proof. This follows from the next to last lemma. □

As is to be expected in view of the existential and universal character of the predicates $<$ and $\le$ on the reals, we have:

Lemma. $x \le y \leftrightarrow y \not< x$.

Proof. $\to$. Assume $x \le y$ and $y < x$. By the previous lemma we obtain $x < x$, a contradiction.

$\leftarrow$. It clearly suffices to show $0 \not< z \to z \le 0$, for a real z given by $(c_n)_n$. Assume $0 \not< z$. We must show $\forall_k \exists_p \forall_{n \ge p} \, c_n \le 2^{-k}$. Let k be given. By assumption $0 \not< z$, hence $\neg \exists_l \, 2^{-l} \le c_{M(l+1)}$, hence $\forall_l \, c_{M(l+1)} < 2^{-l}$. For $l := k+1$ this implies $c_{M(k+2)} < 2^{-k-1}$, hence $c_n \le c_{M(k+2)} + 2^{-k-2} < 2^{-k}$ for $M(k+2) \le n$. □

Constructively, we cannot compare two reals, but we can compare a real with a nontrivial interval ("Approximate Splitting Principle"):

Lemma (ApproxSplit). *Let* x, y, z *be given and assume* $x < y$. *Then either* $z \le y$ *or* $x \le z$.

Proof. Let $x := ((a_n)_n, M)$, $y := ((b_n)_n, N)$, $z := ((c_n)_n, L)$. Assume $x <_k y$, that is (by definition) $1/2^k \le b_p - a_p$ for $p := \max(M(k+2), N(k+2))$. Let $q := \max(p, L(k+2))$ and $d := (b_p - a_p)/4$.

Case $c_q \le \frac{a_p + b_p}{2}$. We show $z \le y$. It suffices to prove $c_n \le b_n$ for $n \ge q$. To see this, observe

$$c_n \le c_q + \frac{1}{2^{k+2}} \le \frac{a_p + b_p}{2} + \frac{b_p - a_p}{4} = b_p - \frac{b_p - a_p}{4} \le b_p - \frac{1}{2^{k+2}} \le b_n.$$

Case $c_q \not\leq \frac{a_p+b_p}{2}$. We show $x \leq z$, via $a_n \leq c_n$ for $n \geq q$.

$$a_n \leq a_p + \frac{1}{2^{k+2}} \leq a_p + \frac{b_p - a_p}{4} \leq \frac{a_p + b_p}{2} - \frac{b_p - a_p}{4} \leq c_q - \frac{1}{2^{k+2}} \leq c_n.$$

This concludes the proof. □

Notice that the boolean object determining whether $z \leq y$ or $x \leq z$ depends on the representation of x, y and z. In particular this assignment is not compatible with our equality relation.

One might think that the non-available comparison of two reals could be circumvented by using a maximum function. Indeed, such a function can easily be defined (component-wise), and it has the expected properties $x, y \leq \max(x, y)$ and $x, y \leq z \to \max(x, y) \leq z$. What is missing is the knowledge that $\max(x, y)$ equals one of its arguments, i.e., we do not have $\max(x, y) = x \vee \max(x, y) = y$.

3 Continuous Functions

We define continuous functions as type-1-objects, and prove their basic properties. This includes proofs of the Intermediate Value Theorem in 3.4 and of the fact that every continuous function with a uniform lower bound on its slope has a continuous inverse, in 3.5.

An inhabited, closed finite interval is called *compact*. We use I, J to denote compact intervals $[a, b]$ with rational end points $a < b$.

3.1 Continuous Functions as Type-1-Objects

Definition. A *continuous function* $f\colon I \to \mathbb{R}$ is given by

(a) an approximating map $h_f\colon (I \cap \mathbb{Q}) \times \mathbb{N} \to \mathbb{Q}$ and a map $\alpha_f\colon \mathbb{N} \to \mathbb{N}$ such that $(h_f(a, n))_n$ is a Cauchy sequence with (uniform) modulus α_f;

(b) a modulus $\omega_f\colon \mathbb{N} \to \mathbb{N}$ of (uniform) continuity, which satisfies

$$|a - b| \leq 2^{-\omega_f(k)+1} \to |h_f(a, n) - h_f(b, n)| \leq 2^{-k} \quad \text{for } n \geq \alpha_f(k);$$

(c) a lower bound N_f and an upper bound M_f for all $h_f(a, n)$.

α_f and ω_f are required to be weakly increasing (that is, $n \leq m \to \alpha_f(n) \leq \alpha)f(m))$.. A function real-valued function defined on an arbitrary interval is continuous if it is continuous on every compact subinterval of with rational end points.

Notice that a continuous function is given by objects of type level ≤ 1 only. This is due to the fact that it suffices to define its values on rational numbers.

The lower and upper bound of the values of the approximating map have been included to ease the definition of composition of continuous functions; however, they also have an effect on computational efficiency.

An example is the *exponential function*. On $[c, d]$ $(c < d)$ it is given by

(a) the approximating map

$$h_{\exp}(a, n) := \sum_{k=0}^{n} \frac{a^k}{k!},$$

and a uniform Cauchy modulus $\alpha_{\exp}$, which can be computed from

$$\Big|\sum_{k=0}^{n} \frac{a^k}{k!} - \sum_{k=0}^{m} \frac{a^k}{k!}\Big| = \Big|\sum_{k=n+1}^{m} \frac{a^k}{k!}\Big| \leq \sum_{k=n+1}^{\infty} \frac{|a|^k}{k!} \leq 2\frac{|a|^{n+1}}{(n+1)!}$$

for $|a| \leq 1 + \frac{n}{2}$ and $n \leq m$;

(b) the modulus of uniform continuity, which can be obtained from

$$\begin{aligned}\Big|\sum_{k=0}^{n} \frac{a^k}{k!} - \sum_{k=0}^{n} \frac{b^k}{k!}\Big| &= \Big|\sum_{k=1}^{n} \frac{a^k - b^k}{k!}\Big| = |a-b| \sum_{k=1}^{n} \frac{1}{k!}\Big|\sum_{l=0}^{k-1} a^{k-1-l} b^l\Big| \\ &\leq |a-b| \sum_{k=1}^{n} \frac{kM^{k-1}}{k!} = |a-b| \sum_{k=0}^{n-1} \frac{M^k}{k!} < |a-b| \exp(M),\end{aligned}$$

where $M = \max(|c|, |d|)$;

(c) the lower bound $N_{\exp} := 0$ and upper bound

$$M_{\exp} := \sum_{k=0}^{L} \frac{d^k}{k!} + 2\frac{|d|^{L+1}}{(L+1)!} \quad \text{with } L := 2\lceil |d| \rceil;$$

these can easily be verified.

3.2 Application of a Continuous Function to a Real

Since the approximating map operates on rationals only, we need to define what it means to apply a continuous function in our sense to a real.

Definition. *Application* of a continuous function $f\colon I \to \mathbb{R}$ (given by h_f, α_f, ω_f) to a real $x := ((a_n)_n, M)$ in I is defined to be

$$(h_f(a_n, n))_n$$

with modulus $\max(\alpha_f(k+2), M(\omega_f(k+1)-1))$. This is a modulus, for

$$\begin{aligned}&|h_f(a_m,m)-h_f(a_n,n)|\\&\le |h_f(a_m,m)-h_f(a_m,p)|+|h_f(a_m,p)-h_f(a_n,p)|+|h_f(a_n,p)-h_f(a_n,n)|\\&\le 2^{-k-2}+2^{-k-1}+2^{-k-2}\end{aligned}$$

if $m,n \ge M(\omega_f(k+1)-1)$ and $p \ge \alpha_f(k+1)$ (for the middle term), and moreover $m,n,p \ge \alpha_f(k+2)$ (for the first and last term). We denote this real by $f(x)$. The set of all such reals is called the *range* of f.

Two continuous functions $f\colon I \to \mathbb{R}$ and $g\colon J \to \mathbb{R}$ are *equal* if $I = J$ and $f(a) = g(a)$ for all rationals $a \in I$.

We show that indeed a continuous function f has ω_f as a modulus of uniform continuity.

Lemma (ContMod). *Let f be continuous and x, y be reals in its domain. Then*

$$|x-y| \le 2^{-\omega_f(k)} \to |f(x)-f(y)| \le 2^{-k}.$$

Proof. Let $x := ((a_n)_n, M)$ and $y := ((b_n)_n, N)$. Assume $|a_n - b_n| \le 2^{-\omega_f(k)+1}$ for $n \ge p$. Then $|h_f(a_n,n) - h_f(b_n,n)| \le 2^{-k}$ for $n \ge p, \alpha_f(k)$, that is $|f(x)-f(y)| \le 2^{-k}$. □

Essentially the same proof shows that application is compatible with equality.

Lemma (ContAppComp). *Let f be a continuous function and x, y be reals in its domain. Then $x = y$ implies $f(x) = f(y)$.*

Proof. Let $x := ((a_n)_n, M)$ and $y := ((b_n)_n, N)$, and assume $x = y$. Given k, pick p such that $|a_n - b_n| \le 2^{-\omega_f(k)+1}$ for $n \ge p$. Then, as in the proof above, $|f(x)-f(y)| \le 2^{-k}$. Hence $f(x) = f(y)$. □

3.3 Continuous Functions and Limits

We show that continuous functions commute with limits.

Lemma (ContLim). *Let $(x_n)_n$ be a sequence of reals which converges to y. Assume $x_n, y \in I$ and let $f\colon I \to \mathbb{R}$ be continuous. Then $(f(x_n))_n$ converges to $f(y)$.*

Proof. For a given k, pick p such that $p \le n \to |x_n - y| \le 2^{-\omega_f(k)}$ for all n. Then by the previous lemma $p \le n \to |f(x_n) - f(y)| \le 2^{-k}$. Hence $(f(x_n))_n$ converges to $f(y)$. □

Lemma (ContRat). *Assume that $f, g\colon I \to \mathbb{R}$ are continuous and coincide on all rationals $a \in I$. Then $f = g$.*

Proof. Let $x := ((a_n)_n, M)$. By ContLim, $(f(a_n))_n$ converges to $f(x)$ and $(g(a_n))_n$ to $g(x)$. Now $f(a_n) = g(a_n)$ implies $f(x) = g(x)$. □

3.4 Intermediate Value Theorem

We next supply the standard constructive versions of the *intermediate value theorem*.

Theorem (Approximate Intermediate Value Theorem). *Let $a < b$ be rational numbers. For every continuous function $f\colon [a,b] \to \mathbb{R}$ with $f(a) \leq 0 \leq f(b)$, and every k, we can find $c \in [a,b]$ such that $|f(c)| \leq 2^{-k}$.*

Proof. In the sequel we repeatedly invoke the Approximate Splitting Principle from 2.5. Given k, let $\varepsilon := 2^{-k}$. We compare $f(a)$ and $f(b)$ with $-\varepsilon < -\frac{\varepsilon}{2}$ and $\frac{\varepsilon}{2} < \varepsilon$, respectively. If $-\varepsilon < f(a)$ or $f(b) < \varepsilon$, then $|f(c)| < \varepsilon$ for $c = a$ or $c = b$; whence we may assume that

$$f(a) < -\frac{\varepsilon}{2} \quad \text{and} \quad \frac{\varepsilon}{2} < f(b).$$

Now pick l so that, for all $x, y \in [a,b]$, if $|x-y| \leq 2^{-l}$, then $|f(x)-f(y)| \leq \varepsilon$, and divide $[a,b]$ into $a = a_0 < a_1 < \cdots < a_m = b$ such that $|a_{i-1} - a_i| \leq 2^{-l}$. Compare every $f(a_i)$ with $-\frac{\varepsilon}{2} < \frac{\varepsilon}{2}$. By assumption $f(a_0) < -\frac{\varepsilon}{2}$ and $\frac{\varepsilon}{2} < f(a_m)$; whence we can find j minimal such that

$$f(a_j) < \frac{\varepsilon}{2} \quad \text{and} \quad -\frac{\varepsilon}{2} < f(a_{j+1}).$$

Finally, compare $f(a_j)$ with $-\varepsilon < -\frac{\varepsilon}{2}$ and $f(a_{j+1})$ with $\frac{\varepsilon}{2} < \varepsilon$. If $-\varepsilon < f(a_j)$, we have $|f(a_j)| < \varepsilon$. If $f(a_{j+1}) < \varepsilon$, we have $|f(a_{j+1})| < \varepsilon$. If both $f(a_j) < -\frac{\varepsilon}{2}$ and $\frac{\varepsilon}{2} < f(a_{j+1})$, then we would have $|f(a_{j+1}) - f(a_j)| > \varepsilon$, contradicting $|a_{j+1} - a_j| \leq 2^{-l}$. □

Alternative Proof. We give a different proof, which more directly makes use of the fact that our continuous functions come with witnessing data.

We may assume $f(a) < -2^{-k-1}$ and $2^{-k-1} < f(b)$ (see above). Divide $[a,b]$ into $a = a_0 < a_1 < \cdots < a_m = b$ such that $|a_{i-1} - a_i| \leq 2^{-\omega_f(k+1)}$. Consider all finitely many

$$h(a_i, n_0) \quad \text{for } i = 1, \ldots, m,$$

with $n_0 := \alpha_f(k+1)$. Pick j such that $h(a_{j-1}, n_0) \leq 0 \leq h(a_j, n_0)$; this can be done because $f(a) < -2^{-k-1}$ and $2^{-k-1} < f(b)$. We show $|f(a_j)| \leq 2^{-k}$; for this it clearly suffices to show $|h(a_j, n)| \leq 2^{-k}$ for $n \geq n_0$. Now

$$|h(a_j,n)| \leq \big|h(a_j,n) - h(a_j,n_0)\big| + \big|h(a_j,n_0)\big| \leq 2^{-k-1} + 2^{-k-1},$$

where the first estimate holds by the choice of n_0, and the second one follows from the choice of a_j and $|h(a_{i-1},n) - h(a_i,n)| \leq 2^{-k-1}$. □

A problem with both of these proofs is that the algorithms they provide are rather bad: in each case one has to partition the interval into as many pieces as the modulus of the continuous function requires for the given error bound, and then for each of these (many) pieces perform certain operations. This problem seems to be unavoidable, since our continuous function may be rather flat. However, we can do somewhat better if we assume a uniform lower bound on the slope of f, that is, some $l \in \mathbb{N}$ such that for all $c, d \in \mathbb{Q}$ and all $m \in \mathbb{N}$

$$\frac{1}{2^m} \leq d - c \to f(c) <_{m+l} f(d).$$

Instead of the linear function $m \mapsto m+l$ one could use (as in [2]) an arbitrary "modulus of increase".

We begin with an auxiliary lemma for the Intermediate Value Theorem, which from a "correct" interval $c < d$ (that is, $f(c) \leq 0 \leq f(d)$ and $2^{-n} \leq d - c$) constructs a new one $c_1 < d_1$ with $d_1 - c_1 = \frac{2}{3}(d - c)$.

Lemma (IVTAux). *Let $f\colon [a, b] \to \mathbb{R}$ be continuous, and with a uniform lower bound l on its slope. Assume $a \leq c < d \leq b$, say $2^{-n} < d - c$, and $f(c) \leq 0 \leq f(d)$. Then we can construct c_1, d_1 with $d_1 - c_1 = \frac{2}{3}(d - c)$, such that again $a \leq c \leq c_1 < d_1 \leq d \leq b$ and $f(c_1) \leq 0 \leq f(d_1)$.*

Proof. Let $c_0 = \frac{2c+d}{3}$ and $d_0 = \frac{c+2d}{3}$. From $2^{-n} < d - c$ we obtain $2^{-n-2} \leq d_0 - c_0$, so $f(c_0) <_{n+2+l} f(d_0)$. Now compare 0 with this proper interval, using ApproxSplit. In the first case we have $0 \leq f(d_0)$; then let $c_1 = c$ and $d_1 = d_0$. In the second case we have $f(c_0) \leq 0$; then let $c_1 = c_0$ and $d_1 = d$. □

Theorem (Intermediate Value Theorem). *If $f\colon [a, b] \to \mathbb{R}$ is continuous with $f(a) \leq 0 \leq f(b)$, and with a uniform lower bound on its slope, then we can find $x \in [a, b]$ such that $f(x) = 0$.*

Proof. Iterating the construction in the auxiliary lemma IVTAux above, we construct two sequences $(c_n)_n$ and $(d_n)_n$ of rationals such that for all n

$$\begin{aligned} &a = c_0 \leq c_1 \leq \cdots \leq c_n < d_n \leq \cdots \leq d_1 \leq d_0 = b, \\ &f(c_n) \leq 0 \leq f(d_n), \\ &d_n - c_n = \big(2/3\big)^n (b - a). \end{aligned}$$

Let x, y be given by the Cauchy sequences $(c_n)_n$ and $(d_n)_n$ with the obvious modulus. As f is continuous, $f(x) = 0 = f(y)$ for the real number $x = y$. □

Remark. The proposition can also be proved for locally nonconstant functions. A function $f\colon [a, b] \to \mathbb{R}$ is *locally nonconstant* whenever if $a \leq a' < b' \leq b$ and c is an arbitrary real, then $f(x) \neq c$ for some real $x \in [a', b']$. Note that if f is continuous, then there also is a rational with that property. Strictly monotonic functions are clearly locally nonconstant, and so are nonconstant real polynomials.

3.5 Inverse Functions

We prove that every continuous function with a uniform lower bound on its slope has a continuous inverse. A constructive proof of this fact has been given by Mandelkern [8]. More recently, J. Berger [2] introduced a concept he called "exact representation of continuous functions", and based on this gave a construction converting an exact representation of an increasing function into an exact representation of its inverse. The proof below is based on our concept of a continuous function as a type-1 object.

Theorem. *Let $f\colon [a,b] \to \mathbb{R}$ be continuous with a uniform lower bound on its slope. Let $f(a) \le a' < b' \le f(b)$. We can find a continuous $g\colon [a',b'] \to \mathbb{R}$ such that $f(g(y)) = y$ for every $y \in [a',b']$ and $g(f(x)) = x$ for every $x \in [a,b]$ such that $a' \le f(x) \le b'$.*

Proof. Let $f\colon [a,b] \to \mathbb{R}$ be continuous with a uniform lower bound on its slope, that is, some $l \in \mathbb{N}$ such that for all $c, d \in [a,b]$ and all $m \in \mathbb{N}$,

$$2^{-m} \le d - c \to f(c) <_{m+l} f(d).$$

Let $f(a) \le a' < b' \le f(b)$. We construct a continuous $g\colon [a',b'] \to \mathbb{R}$.

Let $u \in [a',b']$ be rational. Using $f(a) - u \le a' - u \le 0$ and $0 \le b' - u \le f(b) - u$, the Intermediate Value Theorem gives us an x such that $f(x) - u = 0$, as a Cauchy sequence (c_n). Let $h_g(u,n) := c_n$. Define the modulus α_g such that for $n \ge \alpha_g(k)$, $(2/3)^n(b-a) \le 2^{-\omega_f(k+l+2)}$. For the uniform modulus ω_g of continuity assume $a' \le u < v \le b'$ and $k \in \mathbb{N}$. We claim that with $\omega_g(k) := k + l + 2$ (l from the hypothesis on the slope) we can prove the required property

$$|u - v| \le 2^{-\omega_g(k)+1} \to |h_g(u,n) - h_g(v,n)| \le 2^{-k} \quad (n \ge \alpha_g(k)).$$

Let $a' \le u < v \le b'$ and $n \ge \alpha_g(k)$. For $c_n^{(u)} := h_g(u,n)$ and $c_n^{(v)} := h_g(v,n)$ assume that $|c_n^{(u)} - c_n^{(v)}| > 2^{-k}$; we must show $|u - v| > 2^{-\omega_g(k)+1}$.

By the proof of the Intermediate Value Theorem we have

$$d_n^{(u)} - c_n^{(u)} \le (2/3)^n(b-a) \le 2^{-\omega_f(k+l+2)} \quad \text{for } n \ge \alpha_g(k).$$

Using $f(c_n^{(u)}) - u \le 0 \le f(d_n^{(u)}) - u$, ContMod in 3.2 gives us

$$|f(c_n^{(u)}) - u| \le |(f(d_n^{(u)}) - u) - (f(c_n^{(u)}) - u)| = |f(d_n^{(u)}) - f(c_n^{(u)})| \le 2^{-k-l-2}$$

and similarly $|f(c_n^{(v)}) - v| \le 2^{-k-l-2}$. Hence, using $|f(c_n^{(u)}) - f(c_n^{(v)})| \ge 2^{-k-l}$ (which follows from $|c_n^{(u)} - c_n^{(v)}| > 2^{-k}$ by the hypothesis on the slope),

$$|u - v| \ge |f(c_n^{(u)}) - f(c_n^{(v)})| - |f(c_n^{(u)}) - u| - |f(c_n^{(v)}) - v| \ge 2^{-k-l-1}.$$

Now $f(g(u)) = u$ follows from

$$|f(g(u)) - u| = |h_f(c_n, n) - u| \leq |h_f(c_n, n) - h_f(c_n, m)| + |h_f(c_n, m) - u|,$$

which is $\leq 2^{-k}$ for $n, m \geq \alpha_f(k+1)$. By ContRat, $f(g(y)) = y$ for $y \in [a', b']$.

For every $x \in [a, b]$ with $a' \leq f(x) \leq b'$, from $g(f(x)) < x$ we obtain the contradiction $f(x) = f(g(f(x))) < f(x)$ by the hypothesis on the slope, and similarly for $>$. So we must have $g(f(x)) = x$. □

As an example, consider the squaring function $f\colon [1, 2] \to [1, 4]$, given by the approximating map $h_f(a, n) := a^2$, constant Cauchy modulus $\alpha_f(k) := 1$, modulus $\omega_f(k) := k + 1$ uniform continuity and lower and upper bounds $N_f = 1$ and $M_f = 4$. The lower bound on its slope is $l := 0$, because for all $c, d \in [1, 2]$

$$2^{-m} \leq d - c \to c^2 <_m d^2.$$

Then $h_g(u, n) := c_n^{(u)}$, as constructed in the IVT for $x^2 - u$, iterating IVTAux. The Cauchy modulus α_g is such that $(2/3)^n \leq 2^{-k+3}$ for $n \geq \alpha_g(k)$, and the modulus of uniform continuity is $\omega_f(k) := k + 2$.

4 Program extraction

We now address the issue of extracting computational content of some of these proofs, with machine help. Formalization clearly involves a lot of details, beginning with an appropriate representation of the data. Here we take a direct approach, by explicitly building the required number systems (natural numbers in binary, rationals, reals as Cauchy sequences of rationals with a modulus, continuous functions in the sense of the type-1 representation described above, etc.) It turns out that in this setup extracted programs perform reaonably well.

4.1 Some background on program extraction

Before describing details of the extracted programs we provide some proof-theoretic background on program extraction and its optimization in general and on problems arising in this case study in particular.

The method of program extraction used in this paper is based on *modified realizability* as introduced by Kreisel [7]. In short, from every constructive proof M of a non-Harrop formula A (in natural deduction or a similar proof calculus) one extracts a program $[\![M]\!]$ "realizing" A, essentially, by removing computationally irrelevant parts from the proof (proofs of Harrop formulas have no computational content). The extracted program has some simple type $\tau(A)$ which depends on the logical shape of the proven formula A only. In its original form the extraction process is fairly straightforward, but usually leads to unnecessarily complex programs. In order to obtain

better programs, proof assistants (for instance Coq, Isabelle/HOL, Agda, Nuprl, Minlog) offer various optimizations of program extraction. Below we describe some optimizations implemented in Minlog [10], which are relevant for our present case study.

Animation Suppose a proof of a theorem uses a lemma. Then the proof term contains just the name of the lemma, say `L`. In the term extracted from this proof we want to preserve the structure of the original proof as much as possible, and hence we use a new constant `cL` at those places where the computational content of the lemma is needed. When we want to execute the program, we have to replace the constant `cL` corresponding to a lemma `L` by the extracted program of its proof. This can be achieved by adding computation rules for `cL` and `cGA`. We can be rather flexible here and enable/block rewriting by using `animate`/`deanimate` as desired.

Let It often happens that a subterm has many occurrences in a term, which leads to unwanted recomputations when evaluating it. A possible cure is to "optimize" the term after extraction, and replace for instance $M[x := N]$ with many occurrences of x in M by $(\lambda x M)N$ (or a corresponding "let"-expression). However, this can already be done at the proof level: When an object (value of a variable or realizer of a premise) might be used more than once, make sure (if necessary by a cut) that the goal has the form $A \to B$ or $\forall_x A$. Now use the "identity lemma" `Id`: $\hat{P} \to \hat{P}$, with a predicate variable $\hat{P}$. Its realizer then has the form $\lambda f, x.fx$. If `cId` is not animated, the extracted term has the form $\texttt{cId}(\lambda x M)N$, which is printed as [`let` x N M].

Quantifiers without computational content Besides the usual quantifiers, $\forall$ and $\exists$, Minlog has so-called *non-computational quantifiers*, $\forall^{\mathsf{nc}}$ and $\exists^{\mathsf{nc}}$, which allow for the extraction of simpler programs. The nc-quantifiers, which were first introduced in [3], can be viewed as a refinement of the Set/Prop distinction in constructive type systems like Coq or Agda. Intuitively, a proof of $\forall^{\mathsf{nc}}_x A(x)$ ($A(x)$ non-Harrop) represents a procedure that assigns to every x a proof $M(x)$ of $A(x)$ where $M(x)$ does not make "computational use" of x, i.e., the extracted program $[\![M(x)]\!]$ does not depend on x. Dually, a proof of $\exists^{\mathsf{nc}}_x A(x)$ is proof of $M(x)$ for some x where the witness x is "hidden", that is, not available for computational use. Consequently, the types of extracted programs for nc-quantifiers are $\tau(\forall^{\mathsf{nc}}_{x^\rho} A) = \tau(\exists^{\mathsf{nc}}_{x^\rho} A) = \tau(A)$ as opposed to $\tau(\forall_{x^\rho} A) = \rho \Rightarrow \tau(A)$ and $\tau(\exists_{x^\rho} A) = \rho \times \tau(A)$. The extraction rules are, for example in the case of $\forall^{\mathsf{nc}}$-introduction and -elimination, $[\![(\lambda x.M^{A(x)})^{\forall^{\mathsf{nc}}_x A(x)}]\!] = [\![M]\!]$ and $[\![(M^{\forall^{\mathsf{nc}}_x A(x)} t)^{A(t)}]\!] = [\![M]\!]$ as opposed to $[\![(\lambda x.M^{A(x)})^{\forall_x A(x)}]\!] = [\![\lambda x M]\!]$ and $[\![(M^{\forall_x A(x)} t)^{A(t)}]\!] = [\![Mt]\!]$. In order for the extracted programs to be correct the variable condition for $\forall^{\mathsf{nc}}$-introduction needs to be strengthened by re-

quiring in addition the abstracted variable x not to occur in the extracted program $[\![M]\!]$. Note that for a Harrop formula A the formulas $\forall_x^{\mathsf{nc}} A$ and $\forall_x A$ are equivalent. Similarly, $\exists_x^{\mathsf{nc}} A$ and $\exists_x A$ are equivalent.

Rules Constants – denoting partial computable functionals – are defined via (computation and rewrite) rules (cf. [11] for details). To simplify equational reasoning, the system identifies terms with the same "normal form", and we should be able to add rewrite rules used to generate normal forms. Decidable predicates are viewed as boolean valued functions, so that the rewrite mechanism applies to them as well.

4.2 Program Extraction for the Intermediate Value Theorem

Since the formalized proofs follow rather closely the informal development above, we do not comment on how these proofs are built, but only on the extracted terms.

Here is the extracted term for `ApproxSplit`.

```
(Rec real=>real=>real=>pos=>boole)
([as4,M5]
  (Rec real=>real=>pos=>boole)
  ([as9,M10]
    (Rec real=>pos=>boole)
    ([as13,M14,n15]
      as13(M5(S(S n15))max M10(S(S n15))max M14(S(S n15)))<=
      (as4(M5(S(S n15))max M10(S(S n15)))+
       as9(M5(S(S n15))max M10(S(S n15))))/2)))
```

of type `real=>real=>real=>pos=>boole`. It takes three reals given x, y, z with moduli M, N, K (here given by their Cauchy sequences `as4`, `as9`, `as13` and moduli `M5`, `M10`, `M14`) and a positive number k (here `n15`), and computes $p := \max(M(k+2), N(k+2))$ and $q := \max(p, L(k+2))$. Then the choice whether to go right or left is by computing the boolean value $c_q \leq \frac{a_p + b_p}{2}$.

For the auxiliary lemma `IVTAux` we obtain the extracted term

```
[f0,n1,n2]
 (cId rat@@rat=>rat@@rat)
 ([cd4]
   [let cd5
     ((2#3)*left cd4+(1#3)*right cd4@
      (1#3)*left cd4+(2#3)*right cd4)
     [if (cApproxSplit(RealConstr(f0 approx left cd5)
                                 ([n6]f0 uMod(S(S n6))))
                      (RealConstr(f0 approx right cd5)
                                 ([n6]f0 uMod(S(S n6))))
```

```
    0
    (S(S(n2+n1))))
(left cd4@right cd5)
(left cd5@right cd4)]])
```

of type `cont=>pos=>pos=>rat@@rat=>rat@@rat`. As in the informal proof, it takes a continuous f (here `f0`), a uniform lower bound l on its slope (here `n1`), a positive number n (here `n2`) and two rationals c, d (here the pair `cd4`) such that $2^{-n} < d - c$. Let $c_0 := \frac{2c+d}{3}$ and $d_0 := \frac{c+2d}{3}$ (here the pair `cd5`, introduced via `let` because it is used four times). Then `ApproxSplit` is applied to $f(c_0)$, $f(d_0)$, 0 and the witness $n + 2 + l$ (here `S(S(n2+n1))`) for $f(c_0) < f(d_0)$. In the first case we go left, that is $c_1 := c$ and $d_1 := d_0$, and in the second case we go right, that is $c_1 := c_0$ and $d_1 := d$.

In the proof of the Intermediate Value Theorem, the construction step in `IVTAux` (from a pair c, d to the "better" pair c_0, d_0) had to be iterated, to produce two sequences $(c_n)_n$ and $(d_n)_n$ of rationals. This is the content of a separate lemma `IVTcds`, whose extracted term is

```
[f0,n1,n2](cDC rat@@rat)(f0 doml@f0 domr)
                        ([n4]cIVTAux f0 n1(n2+n4))
```

of type `cont=>pos=>pos=>pos=>rat@@rat`. It takes a continuous $f\colon [a, b] \to \mathbb{R}$ (here `f0`), a uniform lower bound l on its slope (here `n1`), and a positive number k_0 (here `n2`) such that $2^{-k_0} < b - a$. Then the axiom of dependent choice `DC` is used, to construct from an initial pair $(c_0, d_0) = (a, b)$ of rationals (here `f0 doml@f0 domr`) a sequence of pairs of rationals, by iterating the computational content `cIVTAux` of the lemma `IVTAux`.

Now we are ready to formalize the proof of the Intermediate Value Theorem `IVTFinal`. Its extracted term is

```
[f0,n1,n2,n3]RealConstr([n4]left(cIVTcds f0 n1 n2 n4))
                       ([n4]SZero(S(n4+n3)))
```

of type `cont=>pos=>pos=>pos=>real`. It takes a continuous f (here `f0`), a uniform lower bound l on its slope (here `n1`), a positive number k_0 (here `n2`) such that $2^{-k_0} < b - a$, and a positive number k_1 (here `n3`) such that $b - a < 2^{k_1}$. Then the desired zero x of f is constructed as the Cauchy sequence $(c_n)_n$, with a modulus depending on k_1.

To compute approximations of $\sqrt{2}$ we need `RealApprox`, stating that every real can be approximated by a rational. Its extracted term is

```
(Rec real=>pos=>rat)([as2,M3,n4]as2(M3 n4))
```

of type `real=>pos=>rat`. It takes a real x (here given by the Cauchy sequence `as2` and modulus `M3`) and a positive number k (here `n4`), and computes a rational a such that $|x - a| \le 2^{-k}$.

In order to compose `IVTFinal` with `RealApprox`, we prove a proposition `IVTApprox` stating that given an error bound, we can find a rational approximating $\sqrt{2}$ up to this bound. Clearly we need to refer to $\sqrt{2}$ in the statement of the theorem, but on the other hand we do not want to see a representation of $\sqrt{2}$ in the extracted term, but only the construction of the rational approximation from the error bound. Therefore in the statement of `IVTApprox` we use the non-computational quantifier $\exists^{\mathsf{nc}}$, for the zero z of the given continuous f. The extracted term of `IVTApprox` then simply is

```
[f0,n1,n2,n3]cRealApprox(cIVTFinal f0 n1 n2 n3)
```

of type `cont=>pos=>pos=>pos=>pos=>rat`.

Now we "animate" the auxiliary lemmas, that is, unfold all constants with "c" in front of name of the lemma. For `IVTApprox` this gives

```
[f0,n1,n2,n3,n4]
 left
 [let cd5
   ((cDC rat@@rat)(f0 doml@f0 domr)
   ([n5]
     (cId rat@@rat=>rat@@rat)
     ([cd7]
       [let cd8
         ((2#3)*left cd7+(1#3)*right cd7@
          (1#3)*left cd7+(2#3)*right cd7)
         [if (0<=(f0 approx left cd8
                   (f0 uMod(S(S(S(S(S(S(n2+n5+n1)))))))+
                  f0 approx right cd8
                   (f0 uMod(S(S(S(S(S(S(n2+n5+n1)))))))/2)
          (left cd7@right cd8)
          (left cd8@right cd7)]]))
   (PosPred(SZero(S(n4+n3)))))
   [let cd6
    ((2#3)*left cd5+(1#3)*right cd5@
     (1#3)*left cd5+(2#3)*right cd5)
    [if
     (0<=
      (f0 approx left cd6
       (f0 uMod(S(S(S(S(S(S(S(n2+n4+n3+n4+n3+n1)))))))))+
       f0 approx right cd6
       (f0 uMod(S(S(S(S(S(S(S(n2+n4+n3+n4+n3+n1)))))))))/2)
     (left cd5@right cd6)
     (left cd6@right cd5)]]]
```

Let us now use this term to compute approximations of $\sqrt{2}$. First we construct the continuous function $x \mapsto x^2 - 2$ on $[1, 2]$, with its (trivial) uniform

Cauchy modulus and modulus of uniform continuity, and give it the name `a-sq-minus-two`:

```
(define a-sq-minus-two
  (pt "contConstr 1 2([a0,n1]a0*a0-2)([n0]1)S"))
```

We now apply the extracted term of theorem `IVTApprox` to `a-sq-minus-two` and in addition a uniform lower bound l on its slope, a positive number k_0 such that $2^{-k_0} < b - a$, and a positive number k_1 such that $b - a < 2^{k_1}$ (which all happen to be 1 in this case), and normalize the result:

```
(define sqrt-two-approx
  (normalize-term
    (apply mk-term-in-app-form
      (list (proof-to-extracted-term
              (theorem-name-to-proof "IVTApprox"))
            a-sq-minus-two  (pt "1") (pt "1") (pt "1")))))
```

which prints as

```
[n0]
 left[let cd1
       ((cDC rat@@rat)(1@2)
       ([n1]
         (cId rat@@rat=>rat@@rat)
         ([cd3]
           [let cd4
             ((2#3)*left cd3+(1#3)*right cd3@
              (1#3)*left cd3+(2#3)*right cd3)
             [if (0<=(left cd4*left cd4-2+
                      (right cd4*right cd4-2))/2)
              (left cd3@right cd4)
              (left cd4@right cd3)]]))
       (PosPred(SZero(S(S n0)))))
       [let cd2
        ((2#3)*left cd1+(1#3)*right cd1@
         (1#3)*left cd1+(2#3)*right cd1)
        [if (0<=(left cd2*left cd2-2+
                 (right cd2*right cd2-2))/2)
         (left cd1@right cd2)
         (left cd2@right cd1)]]]
```

The term `sqrt-two-approx` has type `pos=>rat`, where the argument k is for the error bound 2^{-k}. We can now directly (that is, without first translating it into a programming language) use it to compute an approximation of $\sqrt{2}$ to say 20 binary digits. To do this, we need to "animate" `Id` and then normalize

the result of applying `sqrt-two-approx` to 20 (we use normalization by evaluation here, for efficiency reasons):

```
(animate "Id")
(pp (nbe-normalize-term-without-eta
      (make-term-in-app-form sqrt-two-approx (pt "20"))))
```

The result (returned in 1.5 seconds) is the rational

```
4642254518592014049 85#32825696739453707 7627
```

or 1.4142135520957329, which differs from $\sqrt{2} = 1.4142135623730951$ at the seventh (decimal) digit.

It should be noted that for efficiency reasons we have equipped our constants for arithmetical operations $+, *, -, /, <$ on rationals with "external code". This means that when the arguments are numerals already, then the result is computed with the (much faster) arithmetic function of the programming language, not by the rules defining the internal constants.

For a further speed-up, we can also translate this internal term (where "internal" means "in our underlying logical language", hence usable in formal proofs) into an expression of a programming language (Scheme in our case). This done by evaluating `(term-to-expr sqrt-two-approx)`:

```
(lambda (n0)
  (car
   (let ([cd1
          (((cdc (cons 1 2))
            (lambda (n1)
              (lambda (cd3)
                (let ([cd4
                       (cons (+ (* 2/3 (car cd3))
                                (* 1/3 (cdr cd3)))
                             (+ (* 1/3 (car cd3))
                                (* 2/3 (cdr cd3))))])
                  (if (<= 0
                          (/ (+ (- (* (car cd4) (car cd4)) 2)
                                (- (* (cdr cd4) (cdr cd4)) 2))
                             2))
                      (cons (car cd3) (cdr cd4))
                      (cons (car cd4) (cdr cd3)))))))
           (pospred (* (+ (+ n0 1) 1) 2)))])
     (let ([cd2
            (cons (+ (* 2/3 (car cd1)) (* 1/3 (cdr cd1)))
                  (+ (* 1/3 (car cd1)) (* 2/3 (cdr cd1))))])
       (if (<= 0
               (/ (+ (- (* (car cd2) (car cd2)) 2)
```

```
                  (- (* (cdr cd2) (cdr cd2)) 2))
               2))
         (cons (car cd1) (cdr cd2))
         (cons (car cd2) (cdr cd1)))))))
```

The result is very close to the internal term displayed above; we have replaced the internal constant `cDC` (computational content of the axiom of dependent choice) by the corresponding Scheme function (a curried form of iteration):

```
(define cdc
  (lambda (init)
    (lambda (step)
      (lambda (n)
        (if (= 1 n)
            init
            ((step n) (((cdc init) step) (- n 1))))))))),
```

the internal arithmetical functions `+`, `*`, `/` by the ones from the programming language and the internal pairing and unpairing functions by `cons`, `car` and `cdr`. – It turns out that this code is reasonably fast: evaluating

```
((ev (term-to-expr sqrt-two-approx)) 20))
```

gives the result in 10 ms.

References

[1] Patrik Andersson. Exact real arithmetic with automatic error estimates in a computer algebra system. Master's thesis, Mathematics department, Uppsala University, 2001.

[2] Josef Berger. Exact calculation of inverse functions. *Math. Log. Quart.*, 51(2):201–205, 2005.

[3] Ulrich Berger. Program extraction from normalization proofs. In M. Bezem and J.F. Groote, editors, *Typed Lambda Calculi and Applications*, volume 664 of *LNCS*, pages 91–106. Springer Verlag, Berlin, Heidelberg, New York, 1993.

[4] Errett Bishop. *Foundations of Constructive Analysis.* McGraw-Hill, New York, 1967.

[5] Errett Bishop. Mathematics as a numerical language. In J. Myhill A. Kino and R.E. Vesley, editors, *Intuitionism and Proof Theory, Proceedings of the summer conference at Buffalo N.Y. 1968*, Studies in logic and the foundations of mathematics, pages 53–71. North–Holland, Amsterdam, 1970.

[6] Luis Cruz-Filipe. *Constructive Real Analysis: a Type-Theoretical Formalization and Applications*. PhD thesis, Nijmegen University, 2004.

[7] Georg Kreisel. Interpretation of analysis by means of constructive functionals of finite types. In A. Heyting, editor, *Constructivity in Mathematics*, pages 101–128. North–Holland, Amsterdam, 1959.

[8] Mark Mandelkern. Continuity of monotone functions. *Pacific J. of Math.*, 99(2):413–418, 1982.

[9] Erik Palmgren. Constructive nonstandard analysis. In A. Petry, editor, *Méthods et analyse non standard*, volume 9, pages 69–97. Cahiers du Centre de Logique, 1996.

[10] Helmut Schwichtenberg. Minlog. In F. Wiedijk, editor, *The Seventeen Provers of the World*, volume 3600 of *LNAI*, pages 151–157. Springer Verlag, 2006.

[11] Helmut Schwichtenberg. Recursion on the partial continuous functionals. To appear: Proc. 2005 ASL European Summer Meeting Athens, 2006.