Optimized program extraction for induction and coinduction

Ulrich Berger and Olga Petrovska

Swansea University, Swansea, SA2 8PP, Wales, UK
{u.berger,o.petrovska}@swansea.ac.uk

Abstract. We prove soundness of an optimized realizability interpretation for a logic supporting strictly positive induction and coinduction. The optimization concerns the special treatment of Harrop formulas which yields simpler extracted programs. We show that wellfounded induction is an instance of strictly positive induction and derive from this a new computationally meaningful formulation of the Archimedean property for real numbers. We give an example of program extraction in computable analysis and show that Archimedean induction can be used to eliminate countable choice.

1 Introduction

This paper studies a constructive logic for strictly positive inductive and coinductive definitions with a realizability interpretation that permits the extraction of programs from proofs in abstract mathematics. Particular attention is paid to a special treatment of Harrop formulas (which have trivial realizers) leading to optimized programs.

Similar work on this topic has been done in [18, 13, 2, 14, 15] and to a large extent implemented in the Minlog system [5]. Related methods of optimized program extraction can be found in [16] and in the systems Coq [9] and Nuprl [11].

Our main contribution is the extension of the realizability interpretation to inductive predicates defined by Harrop operators permitting induction over non-Harrop predicates. This enables us to exhibit wellfounded induction as a special case of strictly positive induction.

We show the usefulness of our results by a simple example in computable analysis, where we identify a new formulation of the Archimedean property as an induction principle and use it to obtain a direct and computationally meaningful proof that the inequality of approximable real numbers implies their apartness. The proof using Archimedean induction is technically and conceptually simpler than the usual proof using the Archimedean property, Markov's principle and the axiom of countable choice. The fact that Archimedean induction can be used to eliminate countable choice makes this principle potentially interesting for constructive mathematics, where one tries to avoid choice principles as far as possible [10, 17].

The theoretical results are to a large extent dual for induction and coinduction. However, our focus is on induction. For applications of coinduction see e.g. [3,

7, 6, 8] and for a discussion of coinduction in type-theoretic systems see [1]. The Soundness Theorem (Thm. 1) shows correctness of the extracted program with respect to a domain-theoretic semantics. Correctness with respect to an operational semantics ensuring, for example, termination of programs is obtained via a Computational Adequacy Theorem which is proven in [2]. Although the logical system and realizability interpretation in [2] is slightly different to ours this does not affect adequacy in our context since programs and their semantics are type free and therefore independent of the logical system.

Our system of strictly positive inductive definitions is proof-theoretically rather strong since it permits nested and interleaved inductive definitions. The proof-theoretic strength of a related system has been analysed in [19], a discussion of this topic can be found in [2].

2 Intuitionistic fixed point logic (IFP)

The language of IFP consists of *formulas* A, B defined simultaneously with *predicates* $\mathcal{P}, \mathcal{Q}$, and *operators* Φ, Ψ. We let $X, Y, \ldots$ range over *predicate variables*, and P, Q over *predicate constants*, each with a fixed arity, s, t range over first-order terms. There is a distinguished 0-ary predicate constant $\bot$ for falsity which we identify with the formula $\bot()$.

$$\begin{array}{lll} \textit{Formulas} \ni A, B & ::= & \mathcal{P}(\vec{t}) \ (\mathcal{P} \text{ not an abstraction, } \vec{t}\,\text{arity}(\mathcal{P}) \text{ many terms}) \\ & & \mid A \wedge B \ \mid \ A \vee B \ \mid \ A \to B \ \mid \ \forall x\ A \ \mid \ \exists x\ A \\ \textit{Predicates} \ni \mathcal{P}, \mathcal{Q} & ::= & X \ \mid \ P \ \mid \ \lambda \vec{x}\ A \ \mid \ \mu\, \Phi \ \mid \ \nu\, \Phi \\ & & (\text{arity}(\lambda \vec{x}\ A) = |\vec{x}|, \ \text{arity}(\mu\Phi) = \text{arity}(\nu\Phi) = \text{arity}(\Phi)) \\ \textit{Operators} \ni \Phi, \Psi & ::= & \lambda X\ \mathcal{P} \quad (\text{arity}(\lambda X\ \mathcal{P}) = \text{arity}(X) = \text{arity}(\mathcal{P})) \end{array}$$

The application of an operator to a predicate is defined as $(\lambda X\, \mathcal{P})(\mathcal{Q}) = \mathcal{P}[\mathcal{Q}/X]$. A definition $\mathcal{P} \stackrel{\text{Def}}{=} \mu\, \Phi$ will also be written $\mathcal{P} \stackrel{\mu}{=} \Phi(\mathcal{P})$ and if $\Phi = \lambda X \lambda \vec{x}\, A$ it may be written $\mathcal{P}(\vec{x}) \stackrel{\mu}{=} A[\mathcal{P}/X]$ (similarly for $\mathcal{P} \stackrel{\text{Def}}{=} \nu\, \Phi$). We use $\equiv$ for equivalence, i.e., $A \equiv B \stackrel{\text{Def}}{=} A \leftrightarrow B$ and $\mathcal{P} \equiv \mathcal{Q} \stackrel{\text{Def}}{=} \forall \vec{x}\, (\mathcal{P}(\vec{x}) \equiv \mathcal{Q}(\vec{x}))$. The bounded quantifiers $\forall x \in \mathcal{P} \ldots$ and $\exists x \in \mathcal{P} \ldots$ abbreviate $\forall x\, (\mathcal{P}(x) \to \ldots)$ and $\exists x\, (\mathcal{P}(x) \wedge \ldots)$.

An expression (formula, predicate, operator) is *strictly positive (s.p.)* in a predicate variable X if it does not contain X free in the premise of an implication. $\lambda Y\, \mathcal{P}$ is *strictly positive* if $\mathcal{P}$ is s.p. in Y. An expression is *regular* if it contains only inductive predicates $\mu\, \Phi$ and coinductive predicates $\nu\, \Phi$ where Φ is s.p.. Throughout the paper it is assumed that *all expressions are regular and all operators mentioned are strictly positive*.

The proof rules of IFP are the usual rules of intuitionistic first-order logic extended by the following rules for inductive and coinductive definitions:

$$\frac{}{\Phi(\mu\, \Phi) \subseteq \mu\, \Phi}\ \text{cl} \qquad \frac{\Phi(\mathcal{P}) \subseteq \mathcal{P}}{\mu\, \Phi \subseteq \mathcal{P}}\ \text{ind}$$

$$\frac{}{\nu\, \Phi \subseteq \Phi(\nu\, \Phi)}\ \text{cocl} \qquad \frac{\mathcal{P} \subseteq \Phi(\mathcal{P})}{\mathcal{P} \subseteq \nu\, \Phi}\ \text{coind}$$

3 Intuitionistic fixed point logic for realizers (RIFP)

The Scott domain of realizers is defined by the recursive domain equation [12]

$$D = \mathbf{Nil} + \mathbf{Lt}(D) + \mathbf{Rt}(D) + \mathbf{Pair}(D \times D) + \mathbf{F}(D \to D)$$

where $+$ denotes the separated sum, $\times$ the Cartesian product and $D \to D$ is the continuous function space. $\mathbf{Nil}, \mathbf{Lt}, \mathbf{Rt}, \mathbf{Pair}, \mathbf{F}$ are mnemonic labels called constructors. The elements of D are of the form $\bot$ (the least element), $\mathbf{Nil}$, $\mathbf{Lt}(d)$, $\mathbf{Rt}(d)$ where $d \in D$, or $\mathbf{F}(f)$ where f is a continuous function from D to D.

Programs denote elements of D. They are defined simultaneously with *function terms* (*functions* for short), which denote continuous functions on D.

$$\begin{array}{rl} \textit{Programs} \ni p, q & ::= a, b \text{ (variables)} \mid \mathbf{Nil} \mid \mathbf{Lt}(p) \mid \mathbf{Rt}(p) \mid \mathbf{Pair}(p, q) \mid \mathbf{F}(\alpha) \\ & \quad \mid \mathbf{case}(p, \alpha, \beta) \mid \mathbf{proj}_i(p) \ (i \in \{0, 1\}) \mid \alpha\ p \mid \mathbf{rec}(\alpha) \\ \textit{Functions} \ni \alpha, \beta & ::= f, g \text{ (variables)} \mid \lambda a\, p \mid \mathbf{app}(p) \end{array}$$

Since Scott domains and continuous functions form a Cartesian closed category and the mapping $(D \to D) \ni f \mapsto \bigsqcup_n f^n(\bot) \in D$ defines a continuous fixed point operator, programs and abstractions have an obvious denotational semantics [12].

To reason formally about realizers and programs we extend IFP by a sort δ for elements of D and a sort $\delta \to \delta$ for continuous functions from D to D. The terms of sort D are the programs, the terms of sort $\delta \to \delta$ are the function terms. We further add predicate variables $\tilde{X}$, $\tilde{Y}$, ... which admit an extra argument of sort δ, and extend the notions of formula, predicate and operator as well as the rules and axioms of IFP accordingly. We add the axioms:

$$\begin{array}{rlrl} \mathbf{case}(\mathbf{Lt}(a), f, g) &= f\, a & \mathbf{app}(\mathbf{F}(f))\, a &= f\, a \\ \mathbf{case}(\mathbf{Rt}(a), f, g) &= g\, a & (\lambda a\, p)\, b &= p[b/a] \quad \text{(for every prog. } p\text{)} \\ \mathbf{proj}_i(\mathbf{Pair}(a_0, a_1)) &= a_i & \mathbf{rec}(f) &= f\, (\mathbf{rec}(f)) \end{array}$$

The resulting system is called *Intuitionistic fixed point logic for realizers* (RIFP).

4 Realizability and Soundness

An expression is *Harrop* if it contains no disjunction or free predicate variable at a s.p. position, it is *non-computational (nc)* if it contains no disjunction or free predicate variable at all. Hence, nc-expressions are Harrop. We assume that to every predicate variable X there is assigned, in a one-to-one fashion, a predicate variable $\tilde{X}$ with one extra argument place for realizers. We define for every expression $\Box$ an expression $\mathbf{R}(\Box)$ (the realizability interpretation of $\Box$) and, if $\Box$ is Harrop, an expression $\mathbf{H}(\Box)$ (a simplified realizability interpretation), more precisely, for a

- formula A a predicate $\mathbf{R}(A)$ with one argument for realizers,
- predicate $\mathcal{P}$ a predicate $\mathbf{R}(\mathcal{P})$ with an extra argument for realizers,
- non-Harrop operator Φ an operator $\mathbf{R}(\Phi)$ with an extra argument for realizers,

- Harrop formula A a formula $\mathbf{H}(A)$,
- Harrop predicate $\mathcal{P}$ a predicate $\mathbf{H}(\mathcal{P})$ of the same arity,
- Harrop operator Φ an operator $\mathbf{H}(\Phi)$ of the same arity.

We sometimes write $a\,\mathbf{r}\,A$ for $\mathbf{R}(A)(a)$ and $\mathbf{r}\,A$ for $\exists a\,a\,\mathbf{r}\,A$. Set $\mathbf{H}_X(\mathcal{P}) \stackrel{\text{Def}}{=} (\mathbf{H}(\mathcal{P}[P/X]))[X/P]$ where P is a fresh predicate constant.

$$
\begin{array}{rll}
a\,\mathbf{r}\,A = \mathbf{H}(A) \wedge a = \mathbf{Nil}\ (A \text{ Harrop}) & & \\
\mathbf{R}(\mathcal{P}) = \lambda(\vec{x}, a)\,(\mathbf{H}(\mathcal{P}) \wedge a = \mathbf{Nil})\ (\mathcal{P} \text{ Harrop}) & & \\
\text{Otherwise} & & \\
a\,\mathbf{r}\,\mathcal{P}(\vec{t}) = \mathbf{R}(\mathcal{P})(\vec{t}, a) & & \mathbf{H}(\mathcal{P}(\vec{t})) = \mathbf{H}(\mathcal{P})(\vec{t}) \\
c\,\mathbf{r}\,(A \wedge B) = \exists a, b\,(c = \mathbf{Pair}(a, b) \wedge a\,\mathbf{r}\,A \wedge b\,\mathbf{r}\,B) & & \mathbf{H}(A \wedge B) = \mathbf{H}(A) \wedge \mathbf{H}(B) \\
\quad (\text{neither } A \text{ nor } B \text{ Harrop}) & & \\
a\,\mathbf{r}\,(A \wedge B) = a\,\mathbf{r}\,A \wedge \mathbf{H}(B) \quad (B \text{ Harrop}) & & \\
b\,\mathbf{r}\,(A \wedge B) = \mathbf{H}(A) \wedge b\,\mathbf{r}\,B \quad (A \text{ Harrop}) & & \\
c\,\mathbf{r}\,(A \vee B) = \exists a\,(c = \mathbf{Lt}(a) \wedge a\,\mathbf{r}\,A \vee c = \mathbf{Rt}(a) \wedge a\,\mathbf{r}\,B) & & \\
c\,\mathbf{r}\,(A \to B) = \exists f\,(c = \mathbf{F}(f) \wedge \forall a\,(a\,\mathbf{r}\,A \to (f\,a)\,\mathbf{r}\,B)) & & \mathbf{H}(A \to B) = \mathbf{r}\,A \to \mathbf{H}(B) \\
\quad (\text{neither } A \text{ nor } B \text{ Harrop}) & & \\
b\,\mathbf{r}\,(A \to B) = \mathbf{H}(A) \to b\,\mathbf{r}\,B \quad (A \text{ Harrop}) & & \\
a\,\mathbf{r}\,\Diamond x\,A = \Diamond x\,(a\,\mathbf{r}\,A) \quad (\Diamond \in \{\forall, \exists\}) & & \mathbf{H}(\Diamond x\,A) = \Diamond x\,\mathbf{H}(A) \\
\mathbf{R}(X) = \tilde{X} & & \mathbf{H}(P) = P \\
\mathbf{R}(\Diamond \Phi) = \Diamond\,\mathbf{R}(\Phi) \quad (\Diamond \in \{\nu, \mu\}) & & \mathbf{H}(\Diamond \Phi) = \Diamond\,\mathbf{H}(\Phi) \\
\mathbf{R}(\lambda\vec{x}\,A) = \lambda\vec{x}\,\mathbf{R}(A) \quad (= \lambda(\vec{x}, a)\,a\,\mathbf{r}\,A) & & \mathbf{H}(\lambda\vec{x}\,A) = \lambda\vec{x}\,\mathbf{H}(A) \\
\mathbf{R}(\lambda X\,\mathcal{P}) = \lambda\tilde{X}\,\mathbf{R}(\mathcal{P}) & & \mathbf{H}(\lambda X\,\mathcal{P}) = \lambda X\,\mathbf{H}_X(\mathcal{P})
\end{array}
$$

It is clear that the operations $\mathbf{R}$ and $\mathbf{H}$ preserve regularity and strict positivity, hence realizability is well-defined. Furthermore, if A is Harrop, then $\mathbf{H}(A) \equiv \mathbf{r}\,A$, and if A is nc, then $\mathbf{H}(A) = A$.

The Soundness Theorem below is restricted to proofs where every instance of induction or coinduction satisfies the condition that either Φ and $\mathcal{P}$ are both Harrop or both non-Harrop or Φ is Harrop and simple (see below) and $\mathcal{P}$ is non-Harrop. We call such proofs *admissible*. This is not a severe restriction since in all practical applications proofs turn out to be admissible. An expression $\Box$ is X*-simple* if no sub-expression of $\Box$ of the form $\mu\,\Phi$ or $\nu\,\Phi$ contains X free. A (strictly positive) operator $\lambda X\,\mathcal{P}$ is *simple* if $\mathcal{P}$ is X-simple. We conjecture that the Soundness Theorem also holds without the admissibility assumption. From now on we tacitly assume that all proofs are admissible.

We write $p\,q$ for $\mathbf{app}(p)\,q$, $\lambda a\,p$ for $\mathbf{F}(\lambda a\,p)$, and $p \circ q$ for $\lambda a\,(p(q\,a))$.

Theorem 1 (Soundness). *Let Γ be a set of Harrop formulas and Δ a set of formulas that are not Harrop. Then, from an admissible IFP-proof of a formula A from the assumptions Γ, Δ one can extract a program p with $\mathrm{FV}(p) \subseteq \vec{u}$ such that $p\,\mathbf{r}\,A$ is RIFP-provable from the assumptions $\mathbf{H}(\Gamma)$ and $\vec{u}\,\mathbf{r}\,\Delta$.*

Proof. By induction on derivations one shows simultaneously

(1) If $\Gamma, \Delta \vdash_{\mathrm{IFP}} A$ where A is not Harrop, then $\mathbf{H}(\Gamma), \vec{u}\,\mathbf{r}\,\Delta \vdash_{\mathrm{RIFP}} p\,\mathbf{r}\,A$ for some program p with $\mathrm{FV}(p) \subseteq \vec{u}$.
(2) If $\Gamma, \Delta \vdash_{\mathrm{IFP}} A$ where A is Harrop, then $\mathbf{H}(\Gamma), \vec{u}\,\mathbf{r}\,\Delta \vdash_{\mathrm{RIFP}} \mathbf{H}(A)$.

The logical rules are easy. To see that the rules for induction and coinduction are realizable in all admissible cases one needs to do a case distinction on whether or not the operator Φ and the predicate $\mathcal{P}$ are Harrop. Note that the case that Φ is non-Harrop but $\mathcal{P}$ is Harrop is excluded due to the admissibility condition.

W.l.o.g. we assume that Φ is not constant; i.e., if $\Phi = \lambda X\, \mathcal{Q}$ then X does occur freely (and hence strictly positively) in $\mathcal{Q}$. We first look at induction

$$\frac{\Phi(\mathcal{P}) \subseteq \mathcal{P}}{\mu\,\Phi \subseteq \mathcal{P}}\ \mathrm{ind}$$

If Φ and $\mathcal{P}$ are both not Harrop, then the premise of induction gives us a realizer s of $\Phi(\mathcal{P}) \subseteq \mathcal{P}$, i.e., $\forall b, \vec{x}\,(b\,\mathbf{r}\,\Phi(\mathcal{P})(\vec{x}) \to (s\,b)\,\mathbf{r}\,\mathcal{P}(\vec{x}))$. Using the notation $g^{-1} \circ \mathcal{Q} \stackrel{\mathrm{Def}}{=} \lambda(\vec{x}, b)\,\mathcal{Q}(\vec{x}, g\,b)$ and the easily provable fact that $\mathbf{R}(\Phi(\mathcal{P})) = \mathbf{R}(\Phi)(\mathbf{R}(\mathcal{P}))$ if Φ and $\mathcal{P}$ are both Harrop, this can be written as

(1) $\mathbf{R}(\Phi)(\mathbf{R}(\mathcal{P})) \subseteq s^{-1} \circ \mathbf{R}(\mathcal{P}))$.

By recursion on the build-up of Φ one can define a closed term $\mathbf{map}$ realizing the formula $X \subseteq Y \to \Phi(X) \subseteq \Phi(Y)$. Using the notation above, this means that for all $g, \tilde{X}, \tilde{Y}$, we have $\tilde{X} \subseteq g^{-1} \circ \tilde{Y} \to \mathbf{R}(\Phi)(\tilde{X}) \subseteq (\mathbf{map}\,g)^{-1} \circ \mathbf{R}(\Phi)(\tilde{Y})$, in particular for $\tilde{X} \stackrel{\mathrm{Def}}{=} g^{-1} \circ \tilde{Y}$ one has

(2) $\mathbf{R}(\Phi)(g^{-1} \circ \tilde{Y}) \subseteq (\mathbf{map}\,g)^{-1} \circ \mathbf{R}(\Phi)(\tilde{Y})$.

We need a realizer of $\mu\,\Phi \subseteq \mathcal{P}$. Since $\mu\,\Phi$ is not Harrop, the realizer f must satisfy $\forall \vec{x} \forall b\,(\mu\,\mathbf{R}(\Phi))(\vec{x}, b) \to (f\,b)\,\mathbf{r}\,\mathcal{P}(\vec{x}))$, i.e., $\mu\,\mathbf{R}(\Phi) \subseteq f^{-1} \circ \mathbf{R}(\mathcal{P})$. We attempt to prove this by induction (with a yet unknown f). Therefore, we try to show $\mathbf{R}(\Phi)(f^{-1} \circ \mathbf{R}(\mathcal{P})) \subseteq f^{-1} \circ \mathbf{R}(\mathcal{P})$.

Using (1) and (2) with $g \stackrel{\mathrm{Def}}{=} f$ and $\tilde{Y} \stackrel{\mathrm{Def}}{=} \mathbf{R}(\mathcal{P})$ we obtain

$$\begin{aligned}\mathbf{R}(\Phi)(f^{-1} \circ \mathbf{R}(\mathcal{P})) &\subseteq (\mathbf{map}\,f)^{-1} \circ \mathbf{R}(\Phi)(\mathbf{R}(\mathcal{P})) \\ &\subseteq (\mathbf{map}\,f)^{-1} \circ (s^{-1} \circ \mathbf{R}(\mathcal{P})) \\ &\equiv (s \circ \mathbf{map}\,f)^{-1} \circ \mathbf{R}(\mathcal{P})\end{aligned}$$

Hence if we define f recursively by $f = s \circ \mathbf{map}\,f$ we are done.

The case that Φ and $\mathcal{P}$ are both Harrop is easy, since then premise and conclusion of the induction rule are Harrop and therefore the realizability interpretation of the premise is $\mathbf{H}(\Phi(\mathcal{P})) \subseteq \mathbf{H}(\mathcal{P})$ and that of the conclusion $\mu\,\mathbf{H}(\Phi) \subseteq \mathbf{H}(\mathcal{P})$. Since one can prove by structural induction that $\mathbf{H}(\Phi(\mathcal{P}))$ is the same as $\mathbf{H}(\Phi)(\mathbf{H}(\mathcal{P}))$ and $\mathbf{H}(\Phi)$ inherits strict positivity from Φ, we obtain an instance of the induction rule for the $\mathbf{H}(\Phi)$ and the $\mathbf{H}(\mathcal{P})$.

The last case to consider is that Φ is Harrop and simple, and $\mathcal{P}$ is not Harrop. The premise of induction gives us a realizer s of $\Phi(\mathcal{P}) \subseteq \mathcal{P}$ i.e., since $\Phi(\mathcal{P})$ is not

Harrop, $\forall b, \vec{x}\,(b\,\mathbf{r}\,\Phi(\mathcal{P})(\vec{x}) \to (s\,b)\,\mathbf{r}\,\mathcal{P}(\vec{x}))$. Using the notation $\mathcal{P}_a \stackrel{\text{Def}}{=} \lambda\vec{x}\,(a\,\mathbf{r}\,\mathcal{P}(\vec{x}))$ this can be written as $\forall b\,(\Phi(\mathcal{P})_b \subseteq \mathcal{P}_{s\,b})$. We need a realizer a of $\mu\,\Phi \subseteq \mathcal{P}$. Since $\mu\,\Phi$ is Harrop, this means that a must satisfy $\forall\vec{x}\,(\mu\,\mathbf{H}(\Phi))(\vec{x}) \to a\,\mathbf{r}\,\mathcal{P}(\vec{x}))$, i.e., $\mu\,\mathbf{H}(\Phi) \subseteq \mathcal{P}_a$. We attempt to prove this by induction (with a yet unknown a). Therefore, we show $\mathbf{H}(\Phi)(\mathcal{P}_a) \subseteq \mathcal{P}_a$. By recursion on the build-up of Φ one can construct a closed (recursion-free) term ψ such that $\mathbf{H}(\Phi)(\mathcal{P}_b) \subseteq \Phi(\mathcal{P})_{\psi(b)}$ for all b. It follows that $\mathbf{H}(\Phi)(\mathcal{P}_a) \subseteq \Phi(\mathcal{P})_{\psi(a)} \subseteq \mathcal{P}_{s\,\psi(a)}$. Hence, if a is defined recursively as $a = s\,\psi(a)$, we are done.

For coinduction the proof is completely dual in the first two cases ($\Phi, \mathcal{P}$ both non-Harrop or both Harrop) and similar to induction in the third case (Φ Harrop, $\mathcal{P}$ non-Harrop).

5 Wellfounded induction

In the following we let upper-case Roman letters range over arbitrary predicates.

The usual formulation of induction over a wellfounded relation $<$ is

$$\frac{\mathbf{Prog}_<(P)}{\forall x\,P(x)}\ \mathbf{WfI}(<)$$

where $\mathbf{Prog}_<(P) \stackrel{\text{Def}}{=} \forall x\,(\forall y\,(y < x \to P(y)) \to P(x))$. In order to be computationally meaningful we formulate this principle in a relativized form where we require the relation $<$ to be wellfounded only on a given predicate A (which is typically non-Harrop). This is expressed by the condition that A is contained in the wellfounded (or accessible) part of $<$. Hence *Wellfounded induction* is the principle

$$\frac{A \subseteq \mathbf{Acc}_< \qquad \mathbf{Prog}_{<,A}(P)}{A \subseteq P}\ \mathbf{WfI}(<, A)$$

where
$$\mathbf{Acc}_<(x) \stackrel{\mu}{=} \forall y < x\,\mathbf{Acc}_<(y)$$
$$\mathbf{Prog}_{<,A}(P) \stackrel{\text{Def}}{=} \forall x \in A\,(\forall y \in A\,(y < x \to P(y)) \to P(x))$$

Proposition 1. *Wellfounded induction follows from admissible s.p. induction. If P is not Harrop, then the extracted program is the least fixed point operator; i.e., if f realizes $\mathbf{Prog}_{<,A}(P)$, then the least fixed point of f realizes $A \subseteq P$.*

Proof. Set $\Phi \stackrel{\text{Def}}{=} \lambda X\,\lambda x\,\forall y < x\,X(y)$ which is a simple Harrop operator. Then $\mathbf{Acc}_< = \mu\,\Phi$ and the assumed progressivity, $\mathbf{Prog}_{<,A}(P)$, is equivalent to $\Phi(A \Rightarrow P) \subseteq (A \Rightarrow P)$, where $A \Rightarrow P \stackrel{\text{Def}}{=} \lambda x\,(A(x) \to P(x))$. Hence $\mu\,\Phi \subseteq (A \Rightarrow P)$, and therefore, by the assumption $A \subseteq \mathbf{Acc}_<$, $A \subseteq (A \Rightarrow P)$. It follows $A \subseteq P$.

Now let P be non-Harrop and let f realize $\mathbf{Prog}_{<,A}(P)$. By the proof of the Soundness Theorem, a defined recursively as $a = f\,\psi(a)$ realizes $\mathbf{Acc}_< \subseteq (A \Rightarrow P)$ where ψ satisfies $\mathbf{H}(\Phi)(Q_b) \subseteq \Phi(Q)_{\psi(b)}$. Unfolding this formula one sees that ψ is the identity. Therefore, the least fixed point of f realizes $\mathbf{Acc}_< \subseteq (A \Rightarrow P)$ and therefore, as can be easily seen, also $A \subseteq P$.

6 Archimedean induction

We give an application of wellfounded induction and hence inductive definitions in computable analysis. We let the variables $x, y, \ldots$ range over abstract reals. We assume that the basic arithmetic operations $(0, 1, +, *, |\cdot|, \ldots)$ and relations $(=, <, \leq, \ldots)$ are given as function and predicate symbols and we will freely use any true arithmetic nc-properties of them. Hence $x = y$, $x < y$, $x \leq y$ are atomic formulas. We write $x \neq y$ as a shorthand for $\neg(x = y)$, i.e. $x = y \to \bot$. All these formulas are nc.

Natural numbers are inductively defined as a subset of the real numbers by

$$\mathbf{N}(x) \stackrel{\mu}{=} (x = 0 \vee \mathbf{N}(x - 1))$$

(i.e. $\mathbf{N} \stackrel{\text{Def}}{=} \mu\,(\lambda X\, \lambda x\, (x = 0 \vee X(x - 1)))$). The formula $\mathbf{N}(x)$ is not Harrop since it contains a disjunction at a strictly positive position. Integers ($\mathbf{Z}$) and rational numbers ($\mathbf{Q}$) are defined from the natural numbers in the usual way.

Cauchy reals are represented as real numbers satisfying the predicate

$$\mathbf{A}(x) \stackrel{\text{Def}}{=} \forall n \in \mathbf{N}\, \exists q \in \mathbf{Q}\, |x - q| \leq 2^{-n}$$

The realizability interpretation of the predicate $\mathbf{N}$ is

$$a\, \mathbf{r}\, \mathbf{N}(x) \stackrel{\mu}{=} a = \mathbf{Lt}(\mathbf{Nil}) \wedge x = 0 \vee \exists b\, (a = \mathbf{Rt}(b) \wedge b\, \mathbf{r}\, \mathbf{N}(x - 1))$$

Hence realizers of the elements of $\mathbf{N}$ are numerals $\mathbf{Rt}^n(\mathbf{Lt}(\mathbf{Nil}))$. We write $\mathbf{0}$ for $\mathbf{Lt}(\mathbf{Nil})$ and $\mathbf{S}(a)$ for $\mathbf{Rt}(a)$.[3] A realizer of $\mathbf{A}(x)$ is a function f such that

$$\forall n, a\, (a\, \mathbf{r}\, \mathbf{N}(n) \to \exists q \in \mathbf{Q}\, (f(a)\, \mathbf{r}\, \mathbf{Q}(q) \wedge |x - q| \leq 2^{-n}))$$

Hence, essentially, f is a sequence of (representations of) rational numbers converging quickly to x. To work in the model of Cauchy reals one simply relativizes all quantifiers to $\mathbf{A}$. However, we refrain from doing so since there are principles (such as Archimedean induction below) which are valid without such relativization.

The usual apartness relation between real numbers is defined by

$$x \mathrel{\#} y \stackrel{\text{Def}}{=} \exists k \in \mathbf{N}\, |x - y| \geq 2^{-k}$$

Clearly, $x \mathrel{\#} 0$ implies $x \neq 0$ but the converse implication only holds with extra assumption on x, for example $x \in \mathbf{A}$. We are interested in a proof of the converse implication that permits the extraction of a program, possibly admitting classically true assumptions as long as they are Harrop or realizable and therefore do not spoil program extraction.

We first prove the implication $x \neq y \to x \mathrel{\#} y$ relativized to $x, y \in \mathbf{A}$ with the help of a Harrop formulation of the Archimedean property, Markov's principle and the countable axiom of choice:

[3] The *binary* representation of natural numbers is obtained by defining the (same) set of natural numbers as $\mathbf{N}(x) \stackrel{\mu}{=} \exists y\, ((y = 0 \vee y > 0 \wedge \mathbf{N}(y)) \wedge \exists i \in \{0, 1\}(x = 2y + i))$.

*Archimedean property (*AP*)*: $(\forall n \in \mathbf{N}\, |x| < 2^{-n}) \to x = 0$.
*Markov's principle (*MP*)*:
$\forall n \in \mathbf{N}\, (A(n) \vee \neg A(n)) \to \neg\neg \exists n \in \mathbf{N}\, A(n) \to \exists n \in \mathbf{N}\, A(n)$
*Axiom of countable choice (*ACC*)*:
$\forall n \in \mathbf{N}\, \exists x\, A(n,x) \to \exists f\, \forall n \in \mathbf{N}\, A(m, f(n))$.

Note that AP is a Harrop formula which is equivalent to $\mathbf{H}(\mathrm{AP})$. MP is realized by an unbounded search operator which can be easily defined by recursion. ACC quantifies over functions, hence requires an extension of IFP, and is realized by the identity.

Proposition 2 $(\mathrm{AP}, \mathrm{MP}, \mathrm{ACC})$**.** $\forall x \in \mathbf{A}\, (x \neq 0 \to x \not\equiv 0)$.

Proof. Assume $\mathbf{A}(x)$ and $x \neq 0$. By ACC there exists an infinite sequence of rational numbers q_k $(k \in \mathbf{N})$ such that $|x - q_k| \leq 2^{-k}$ for all $k \in \mathbf{N}$. It is impossible that $|q_{k+1}| \leq 2^{-k}$ for all $k \in \mathbf{N}$ since this would clearly imply that $|x| \leq 2^{-k}$ for all $k \in \mathbf{N}$ and therefore $x = 0$, by AP. Since $|q_{k+1}| \leq 2^{-k}$ is a decidable property of k, by MP we can find some $k \in \mathbf{N}$ with $|q_{k+1}| > 2^{-k}$. It follows that $|x| \geq 2^{-(k+1)}$.

We now introduce an alternative formulation of the Archimedean property in the form of an induction principle. This will allow us to prove the implication $x \neq 0 \to x \not\equiv 0$ for $x \in \mathbf{A}$ without using Markov's principle or countable choice and will directly yield a simple extracted program.

Archimedean induction is the rule

$$\frac{\forall x \neq 0\, ((|x| \leq 3 \to P(2x)) \to P(x))}{\forall x \neq 0\, P(x)}\ \mathrm{AI}$$

Of course, the number 3 can be replaced by any positive rational number and the number 2 by any rational number > 1.

Proposition 3. *AI follows classically from AP and wellfounded induction. If P is not Harrop, then AI is realized by* **rec**.

Proof. By the Archimedean property, for each $x \neq 0$, the sequence $|x|, |2x|, |4x|, \ldots$ is unbounded, hence will eventually exceed 3. Therefore, $A \subseteq \mathbf{Acc}_{\prec}$ holds, where $A(x) \stackrel{\mathrm{Def}}{=} x \neq 0$ and $y \prec x \stackrel{\mathrm{Def}}{=} |x| \leq 3 \wedge y = 2x$. The premise of AI is $\mathbf{Prog}_{\prec,A}(P)$, hence $A \subseteq P$, by $\mathbf{WfI}(\prec, A)$. By Prop. 1 the extracted realizer is **rec**.

A useful variant of Archimedean induction is its relativization to $\mathbf{A}$:

$$\frac{\forall x \in \mathbf{A} \setminus \{0\}\, ((|x| \leq 3 \to P(2x)) \to P(x))}{\forall x \in \mathbf{A} \setminus \{0\}\, P(x)}\ \mathrm{AIC}$$

Proposition 4. AIC *follows from* AI *and hence is realizable.*

Proof. Apply AI to the predicate $\mathbf{A} \Rightarrow P \stackrel{\mathrm{Def}}{=} \lambda x\, (\mathbf{A}(x) \to P(x))$ and use the fact that $\mathbf{A}$ is closed under doubling.

If s realizes the premise of AIC, then a realizer of the conclusion of AIC is extracted as the recursively defined function $\chi\, g = s\, g\, (\chi\, (d\, g))$ where $d = \lambda g\, \lambda n\ 2{*}(g(\mathbf{S}(n)))$ is the realizer extracted from the easy proof of $\mathbf{A}(x) \to \mathbf{A}(2x)$ and $2*$ implements doubling of (unary representations of) natural numbers.

Proposition 5 (AIC)**.** $\forall x \in \mathbf{A}\, (x \neq 0 \to x \not\equiv 0)$.

Proof. We show $\forall x \in \mathbf{A} \setminus \{0\}\, x \not\equiv 0$ using AIC. Let $x \in \mathbf{A} \setminus \{0\}$ and assume, as induction hypothesis, $|x| \leq 3 \to 2x \not\equiv 0$. Since $x \in \mathbf{A}$ there is $q \in \mathbf{Q}$ such that $|x - q| \leq 1$. If $|q| > 2$, then $|x| \geq 1$ and we are done. If $|q| \leq 2$, then $|x| \leq 3$ so we can apply the induction hypothesis to obtain $2x \not\equiv 0$, which implies $x \not\equiv 0$.

Program extraction for Prop. 5: The program extracted from above proof is

$$\varphi\, f = \mathbf{if}\, |f\, \mathbf{0}| > \mathbf{2}\, \mathbf{then}\, \mathbf{1}\, \mathbf{else}\, \mathbf{S}(\varphi\, (\lambda n\, \mathbf{2} * f(\mathbf{S}(n))))$$

where $|\cdot|$ and $>$ implement the absolute value function and $>$ relation on (representations of) rational numbers and $\mathbf{if}\, t\, \mathbf{then}\, p\, \mathbf{else}\, q$ stands for $\mathbf{case}(t, \lambda a\, p, \lambda a\, q)$ assuming that the Booleans are encoded as $\mathbf{Lt}(\mathbf{Nil})$ and $\mathbf{Rt}(\mathbf{Nil})$.

7 Conclusion

We presented a constructive theory of strictly positive inductive and coinductive definitions that permits (co)induction over a Harrop operator to be applied to non-Harrop predicates. This allowed us to exhibit wellfounded induction as a special case of strictly positive induction and, in turn, to give a new presentation of the Archimedean property for real numbers as a computationally meaningful induction principle. A simple example in computable analysis reveals that Archimedean induction is able to provide a new computationally meaningful proof (Prop. 5) of a result that would normally be proven using countable choice plus Markov's principle (Prop. 2). Hence Archimedean induction allowed us to eliminate these constructively questionable principles. We leave it as an open question whether for this particular example (approximable non-zero reals are apart from 0) a computationally meaningful proof using the Archimedean property and Markov's principle alone could be given, but we conjecture that this is not the case. Archimedean induction (even with Markov's principle) is weaker than countable choice since the computable reals validate the former while, classically, they do not validate the latter. Regarding the constructive status of Archimedean induction it must be noted that its reduction to wellfounded induction (Prop. 3) uses classical logic but no choice. Hence this achieves only classical elimination of choice. However, there might be an independent constructive justification of Archimedean induction. At least computationally this principle *is* justified through its realizability interpretation.

It is straightforward to extend our results to generally positive inductive and coinductive definitions. There is even a possibility of extending this to a system of higher-order logic and monotone inductive and coinductive definition as presented in [4].

Acknowledgments. This work was supported by the Marie Curie International Research Staff Exchange Schemes *Computable Analysis* (PIRSES-GA-2011-294962) and *Correctness by Construction* (FP7-PEOPLE-2013-IRSES-612638) as well as the Marie Curie RISE project *Computing with Infinite Data* (H2020-MSCA-RISE-2016-731143) and the EPSRC Doctoral Training Grant No. 1818640.

References

1. A. Abel, B. Pientka, D. Thibodeau, and A. Setzer. Copatterns: Programming infinite structures by observations. In *40th ACM SIGPLAN-SIGACT Symposium on Principles of Prog. Languages (POPL'13)*, pages 27–38, 2013.
2. U. Berger. Realisability for induction and coinduction with applications to constructive analysis. *Jour. Universal Comp. Sci.*, 16(18):2535–2555, 2010.
3. U. Berger. From coinductive proofs to exact real arithmetic: theory and applications. *Logical Methods in Comp. Sci.*, 7(1):1–24, 2011. Paper 8, Pub. 24.03.2011.
4. U. Berger and T. Hou. A realizability interpretation of Church's simple theory of types. *Math. Structures in Comp. Sci.*, pages 1–22, 2016.
5. U. Berger, K. Miyamoto, H. Schwichtenberg, and M. Seisenberger. Minlog - a tool for program extraction for supporting algebra and coalgebra. In *CALCO-Tools*, volume 6859 of *LNCS*, pages 393–399. Springer, 2011.
6. U. Berger, K. Miyamoto, H. Schwichtenberg, and H. Tsuiki. Logic for Gray-code computation. In *Concepts of Proof in Math., Phil., & Comp. Sci.* de Gruyter, 2016.
7. U. Berger and M. Seisenberger. Proofs, programs, processes. *Theory of Computing Systems*, 51(3):213–329, 2012.
8. U. Berger and D. Spreen. A coinductive approach to computing with compact sets. *Journal of Logic and Analysis*, 8, 2016.
9. Y. Bertot and P. Castéran. *Interactive theorem proving and program development: Coq'Art: the Calculus of Inductive Constructions*. Springer, 2004.
10. D. Bridges, F. Richman, and P. Schuster. Linear independence without choice. *Annals of Pure and Applied Logic*, 101(1):95–102, 1999.
11. R. Constable. *Implementing Mathematics with the Nuprl Proof Development System*. Prentice–Hall, New Jersey, 1986.
12. G. Gierz, K. Hofmann, K. Keimel, J. Lawson, M. Mislove, and D. Scott. *Continuous Lattices and Domains*. Encyclop. of Math. and its Applications. CUP 93, 2003.
13. F. Miranda-Perea. Realizability for monotone clausular (co)inductive definitions. *Electr. Notes in Theoret. Comp. Sci.*, 123:179–193, 2005.
14. K. Miyamoto. *Program Extraction from Coinductive Proofs and its Application to Exact Real Arithmetic*. PhD thesis, Mathematisches Institut LMU, Munich, 1993.
15. K. Miyamoto, F. Forsberg, and H. Schwichtenberg. Program Extraction from Nested Definitions. In S. Blazy, C. Paulin-Mohring, and D. Pichardie, editors, *Proc. of the 4th ITP Conf.*, number 7988 in LNCS, pages 370–385. Springer, 2013.
16. M. Parigot. Recursive programming with proofs. *Theoretical Comp. Sci.*, 94(2):335–356, 1992.
17. F. Richman. The fundamental theorem of algebra: a constructive development without choice. *Pacific Journal of Math.*, 196(1):213–230, 2000.
18. M. Tatsuta. Realizability of monotone coinductive definitions and its application to program synthesis. In R. Parikh, editor, *Math. of Prog. Const.*, volume 1422 of *Lecture Notes in Math.*, pages 338–364. Springer, 1998.
19. S. Tupailo. On the intuitionistic strength of monotone inductive definitions. *The Journal of Symbolic Logic*, 69(3):790–798, 2004.