Realisability and adequacy for (co)induction

Ulrich Berger

Swansea University, Swansea, SA2 8PP, Wales UK
u.berger@swansea.ac.uk

Abstract. We prove the correctness of a formalised realisability interpretation of extensions of first-order theories by inductive and coinductive definitions in an untyped λ-calculus with fixed-points. We illustrate the use of this interpretation for program extraction by some simple examples in the area of exact real number computation, and hint at further non-trivial applications in computable analysis.

1 Introduction

This paper studies a formalised realisability interpretation of an extension of first-order predicate logic by least and greatest fixed points of strictly positive operators. The main technical results are the *Soundness Theorem* for this interpretation and the *Adequacy Theorem* for the realisers with respect to a call-by-name operational semantics and a domain-theoretic denotational semantics. Both results together imply the *Program Extraction Theorem* stating that from a constructive proof one can extract a program that is *provably correct* (by the Soundness Theorem) and *terminating* (by the Adequacy Theorem).

In order to get a flavour of the system we discuss some examples within the first-order theory of real closed fields with the real numbers as intended model. In the first example we define a set $\mathbb{N}$ of real numbers (inductively) as the *least* set satisfying

$$\mathbb{N}(0) \wedge \forall x\, (\mathbb{N}(x) \to \mathbb{N}(x + 1))$$

More formally, $\mathbb{N} := \mu X.\{x \mid x = 0 \vee \exists y\, (x = y + 1 \wedge X(y))\}$, i.e. $\mathbb{N}$ is the *least fixed point* of the operator mapping a set X to the set $\{x \mid x = 0 \vee \exists y\, (x = y + 1 \wedge X(y))\}$. Clearly, in the intended model $\mathbb{N}$ is the set of natural numbers.

For the second example, set $\mathbb{I} := [-1, 1] = \{x \mid -1 \leq x \leq 1\}$, $\mathrm{SD} := \{0, 1, -1\}$, and $\mathrm{av}_i(x) := (x + i)/2$. Define C_0 (coinductively) as the *largest* set of real numbers satisfying

$$\forall x\, (\mathrm{C}_0(x) \to \exists i \in \mathrm{SD}, y \in \mathbb{I}\, (x = \mathrm{av}_i(y) \wedge \mathrm{C}_0(y)))$$

Formally, $\mathrm{C}_0 := \nu X.\{x \mid \exists i \in \mathrm{SD}, y \in \mathbb{I}\, (x = \mathrm{av}_i(y) \wedge X(y))\}$, i.e. C_0 is the *greatest fixed point* of the operator mapping X to $\{x \mid \exists i \in \mathrm{SD}, y \in \mathbb{I}\, (x = \mathrm{av}_i(y) \wedge X(y))\}$. One easily shows that $\mathrm{C}_0 = \mathbb{I}$. Hence the coinductive definition seems to be unnecessary. However, the point is that in order to prove constructively $\mathrm{C}_0(x)$ for $x \in \mathbb{I}$, one needs the extra assumption that there is a

rational Cauchy sequence converging to x, and the (coinductive) proof gives us a (coiterative) *program* transforming the Cauchy sequence into a signed digit representation of x.

Our third example extends the previous one to unary functions. Add a new sort for real functions, and let $\mathbb{I}^{\mathbb{I}}$ denote the set of real functions mapping $\mathbb{I}$ to $\mathbb{I}$. Define a set of real functions by

$$\mathrm{C}_1 := \nu F.\mu G.\{g \mid \exists i \in \mathrm{SD}, f \in \mathbb{I}^{\mathbb{I}} \, (g = \mathrm{av}_i \circ f \wedge F(f)) \vee \forall i \in \mathrm{SD} \, G(g \circ \mathrm{av}_i)\}$$

One can show that C_1 coincides with the set of functions in $\mathbb{I}^{\mathbb{I}}$ that are (constructively) uniformly continuous on $\mathbb{I}$. Moreover, a constructive proof of $\mathrm{C}_1(f)$ contains a program that implements f as a non-wellfounded tree acting as a (signed digit) stream transformer similar to the structures studied by Ghani, Hancock and Pattinson [GHP06]. More precisely, this interpretation is the computational content of a constructive proof the formula $\forall f \, (\mathrm{C}_1(f) \to \forall x \, (\mathrm{C}_0(x) \to \mathrm{C}_0(f(x))))$, which is a special case of a constructive composition theorem for analogous predicates C_n of n-ary functions. Details as well as concrete applications with extracted Haskell programs are worked out in [Ber].

Note that in the definition of C_1, the inner inductive definition depends on the set parameter F which is then maximised in the outer coinductive definition. In the context of classical propositional modal logic a system allowing similar "interleaved" least and largest fixed points is known as the *μ-calculus* [BS07b]. Möllerfeld [Mö3] studied the first-order version of the μ-calculus (which is essentially the classical version of our system) and showed that it has the same proof-theoretic strength as Π_2^1-comprehension. Tupailo [Tup04] later showed that the intuitionistic version has the same strength. Möllerfeld used iterated interleavings of least and greatest fixed points to define generalisations of the Souslin quantifier allowing the emulation of non-monotonic inductive definitions which lead to this enormous strength. If one forbids these interleavings, one obtains the proof-theoretically much weaker system $\mathrm{ID}^{<\omega}$ of *finitely iterated inductive definitions* [BFPS81].

The realisability interpretation we are going to study is related to interpretations given by Tatsuta [Tat98] and Miranda-Perea [MP05]. We try to point out the main similarities and differences. Like Tatsuta, we use *untyped* programs as realisers that allow for unrestricted recursion. The necessary termination proof for extracted programs (which seems to be missing in Tatsuta's paper) is obtained by a general Adequacy Theorem relating the operational with a (domain-theoretic) denotational semantics. On the other hand, Miranda extracts typed terms and uses the more general "Mendler-style" (co)inductive definitions [Men91] which extract strongly normalising terms in extensions of the second-order polymorphic λ-calculus or stronger systems [Mat01,AMU05]. Furthermore, Tatsuta studies realisability with truth while we omit the "truth" component. From a practical point of view the most important difference to Tatsuta's interpretation is that we treat quantifiers uniformly in the realisability interpretation (as Miranda-Perea does): $M \, \mathbf{r} \, \forall x \, A(x)$ is defined as $\forall x \, (M \, \mathbf{r} \, A(x))$ (but not $\forall x \, (M \, x \, \mathbf{r} \, A(x))$) and $M \, \mathbf{r} \, \exists x \, A(x)$ is defined as $\exists x \, (M \, \mathbf{r} \, A(x))$ (but not

$\pi_2(M) \mathbf{r} A(\pi_1(M)))$. In general, a realiser never depends on variables of the object language and does not produce output in that language, i.e. the object language and the language of realisers are kept strictly separate. Realisers are extracted exclusively from the "propositional skeleton" of a proof ignoring the first-order part which matters for the *correctness* of the realisers only. This widens the scope of applications considerably because it is now possible to deal with abstract structures that are not necessarily "constructively" given. For example the real numbers in our examples above, were treated abstractly (i.e. axiomatically) without assuming them to be constructed in a particular way. The ignorance w.r.t. the first-order part can also be seen as a special case of the interpretations studied by Schwichtenberg [Sch08], Hernest and Oliva [HO08] and Ratiu and Trifonov [RT09], which allow for a fine control of the amount of computational information extracted from proofs.

2   Induction and coinduction

We fix a first-order language $\mathcal{L}$. *Terms*, $r, s, t \ldots$, are built from constants, first-order variables and function symbols as usual. *Formulas*, $A, B, C \ldots$, are $s = t$, $\mathcal{P}(\boldsymbol{t})$ where $\mathcal{P}$ is a predicate (predicates are defined below), $A \wedge B$, $A \vee B$, $A \to B$, $\forall x\, A$, $\exists x\, A$. A *predicate* is either a predicate constant P, or a predicate variable X, or a comprehension term $\lambda \boldsymbol{x}.A$ (sometimes also written $\{\boldsymbol{x} \mid A\}$) where A is a formula and $\boldsymbol{x}$ is a vector of first-order variables, or an inductive predicate $\mu X.\mathcal{P}$, or a coinductive predicate $\nu X.\mathcal{P}$ where $\mathcal{P}$ is a predicate of the same arity as the predicate variable X and which is *strictly positive* in X, i.e. X does not occur free in any premise of a subformula of $\mathcal{P}$ which is an implication. The application, $\mathcal{P}(\boldsymbol{t})$, of a predicate $\mathcal{P}$ to a list of terms $\boldsymbol{t}$ is a primitive syntactic construct, except when $\mathcal{P}$ is a comprehension term, $\mathcal{P} = \{\boldsymbol{x} \mid A\}$, in which case $\mathcal{P}(\boldsymbol{t})$ stands for $A[\boldsymbol{t}/\boldsymbol{x}]$.

It will sometimes be convenient to write $\boldsymbol{x} \in \mathcal{P}$ instead of $\mathcal{P}(\boldsymbol{x})$ and also $\mathcal{P} \subseteq \mathcal{Q}$ for $\forall \boldsymbol{x}\,(\mathcal{P}(\boldsymbol{x}) \to \mathcal{Q}(\boldsymbol{x}))$ and $\mathcal{P} \cap \mathcal{Q}$ for $\{\boldsymbol{x} \mid \mathcal{P}(\boldsymbol{x}) \wedge \mathcal{Q}(\boldsymbol{x})\}$, e.t.c. We also write $\{t \mid A\}$ as an abbreviation for $\{x \mid \exists \boldsymbol{y}\,(x = t \wedge A)\}$ where x is a fresh variable and $\boldsymbol{y} = \mathrm{FV}(t) \cap \mathrm{FV}(A)$. Furthermore, we introduce *operators* $\Phi := \lambda X.\mathcal{P}$ (or $\Phi(X) := \mathcal{P}$), where $\mathcal{P}$ is strictly positive in X, and then write $\Phi(\mathcal{Q})$ for the predicate $\mathcal{P}[\mathcal{Q}/X]$ where the latter is the usual substitution of the predicate $\mathcal{Q}$ for the predicate variable X. We also write $\mu\Phi$ and $\nu\Phi$ for $\mu X.\mathcal{P}$ and $\nu X.\mathcal{P}$. For convenience, we also write $A(X)$ to distinguish a particular predicate variable X in A, and $A(\mathcal{P})$ for the substitution of every free occurrence of X in A by $\mathcal{P}$. A formula, predicate, or operator is called *non-computational*, if it contains neither free predicate variables nor the propositional connective $\vee$. Otherwise it is called *computational*.

The *proof rules* are the usual rules of intuitionistic predicate calculus with equality augmented by rules expressing that $\mu\Phi$ and $\nu\Phi$ are the least and greatest fixed points of the operator Φ. As is well-known, the fixed point property can be replaced by appropriate inclusions. Hence we stipulate the axioms and rules

$$\text{Closure} \quad \Phi(\mu\Phi) \subseteq \mu\Phi \qquad \text{Induction} \quad \Phi(\mathcal{Q}) \subseteq \mathcal{Q} \to \mu\Phi \subseteq \mathcal{Q}$$

$$\text{Coclosure} \ \nu\Phi \subseteq \Phi(\nu\Phi) \qquad \text{Coinduction} \ \mathcal{Q} \subseteq \Phi(\mathcal{Q}) \to \mathcal{Q} \subseteq \nu\Phi$$

In addition we allow any axioms expressible by non-computational formulas that hold in the intended model. We write $\Gamma \vdash A$ if A is derivable from assumptions in Γ in this system. If A is derivable without assumptions we write $\vdash A$, or even just A. We define falsity as $\bot := \mu X.X$ where X is a 0-ary predicate variable (i.e. a propositional variable). From the induction axiom for $\bot$ it follows immediately $\bot \to A$ for every formula A.

Lemma 1 (Instantiation). *If $\Gamma(X) \vdash A(X)$, then $\Gamma(\mathcal{P}) \vdash A(\mathcal{P})$.*

Proof. Straightforward induction on derivations.

Lemma 2 (Monotonicity). *Let Φ, Ψ be operators, $\mathcal{P}$, $\mathcal{Q}$ predicates, Γ a context and X a predicate variable not free in Γ.*

(a) If $\Gamma \vdash \Phi(X) \subseteq \Psi(X)$, then $\Gamma \vdash \mu\Phi \subseteq \mu\Psi$ and $\Gamma \vdash \nu\Phi \subseteq \nu\Psi$.
(b) $\mathcal{P} \subseteq \mathcal{Q} \to \Phi(\mathcal{P}) \subseteq \Phi(\mathcal{Q})$.

Proof. (a) Assume Γ. We show $\mu\Phi \subseteq \mu\Psi$ using the Induction Axiom. Hence, we have to show $\Phi(\mu\Psi) \subseteq \mu\Psi$. By the hypothesis of the lemma, the Instantiation Lemma, and the Closure Axiom, we have $\Phi(\mu\Psi) \subseteq \Psi(\mu\Psi) \subseteq \mu\Psi$. The proof for ν is similar.

(b) Straightforward induction on the built-up of Φ, using (a) in the case of inductive and coinductive predicates.

Lemma 3 (Fixed Point). *Let Φ be an operator.*

(a) $\Phi(\mu\Phi) = \mu\Phi$.
(b) $\Phi(\nu\Phi) = \nu\Phi$.

Proof. Because of the closure axiom, it suffices for (a) to show $\mu\Phi \subseteq \Phi(\mu\Phi)$. We use the induction rule. Thus it suffices to show $\Phi(\Phi(\mu\Phi)) \subseteq \Phi(\mu\Phi)$. But this follows from the Closure Axiom and the Monotonicity Lemma. The proof of (b) is similar.

3 Realisability

The realisers of formulas are terms of a LISP-like untyped λ-calculus with pairing, injections and recursion (which in Sect. 5 will however receive a call-by-name operational semantics). *Program-terms*, $M, N, K, L, R \ldots$ (*terms* for short) are variables $x, y, z, \ldots$, the constant (), and the composite terms $\langle M, N \rangle$, $\mathrm{inl}(M)$, $\mathrm{inr}(M)$, $\lambda x.M$, $\pi_i(M)$ $(i = 1, 2)$, $\mathrm{case}\, M \,\mathrm{of}\{\mathrm{inl}(x) \to L \,;\, \mathrm{inr}(y) \to R\}$, $(M\, N)$, $\mathrm{rec}\, x \,.\, M$. The *free variables* of a term are defined as usual (the constructs λx, $\mathrm{rec}\, x$ and $\mathrm{inl}(x) \to$, $\mathrm{inr}(x) \to$ in a case term bind the variable x). The usual conventions concerning bound variables apply.

Of particular interest are closed terms that are built exclusively from () by pairing $\langle \cdot, \cdot \rangle$ and the injections $\text{inl}(\cdot)$ $\text{inr}(\cdot)$. We call these terms *data* and denote them by $d, e, \ldots$. Roughly speaking, data stand for themselves and will in any reasonable denotational semantics coincide with their value. In Section 5 we will study such a denotational and also an operational semantics for arbitrary program terms and prove an Adequacy Theorem.

In order to formalise realisability we need a system that can talk about mathematical objects *and* realisers. Therefore we extend our first-order language $\mathcal{L}$ to a language $\mathbf{r}(\mathcal{L})$ by adding a new sort for program terms. All logical operations, including inductive and coinductive definitions, are extended as well. All axioms and rules for $\mathcal{L}$, including closure, induction, coclosure and coinduction and the rules for equality, are extended mutatis mutandis for $\mathbf{r}(\mathcal{L})$. In addition, we have as extra axioms the equations

$$\pi_i(\langle M_1, M_2 \rangle) = M_i \qquad (i = 1, 2)$$
$$\text{case}\,\text{inl}(M)\,\text{of}\{\text{inl}(x) \to L\,;\,\text{inr}(y) \to R\} = L[M/x]$$
$$\text{case}\,\text{inr}(M)\,\text{of}\{\text{inl}(x) \to L\,;\,\text{inr}(y) \to R\} = R[M/y]$$
$$(\lambda x.M)N = M[N/x]$$
$$\text{rec}\,x\,.\,M = M[\text{rec}\,x\,.\,M/x]$$

The realisability interpretation assigns to every $\mathcal{L}$-formula A a unary $\mathbf{r}(\mathcal{L})$-predicate $\mathbf{r}(A)$. Intuitively, for any program term M the $\mathbf{r}(\mathcal{L})$-formula $\mathbf{r}(A)(M)$ (sometimes also written $M\,\mathbf{r}\,A$) states that M "realises" A. The definition of $\mathbf{r}(A)$ is relative to a fixed one-to-one mapping from $\mathcal{L}$-predicate variables X to $\mathbf{r}(\mathcal{L})$-predicate variables $\tilde{X}$ with one extra argument place for program terms. The definition of $\mathbf{r}(A)$ is such that if the formula A has the free predicate variables $X_1, \ldots, X_n$, then the predicate $\mathbf{r}(A)$ has the free predicate variables $\tilde{X}_1, \ldots, \tilde{X}_n$. Simultaneously with $\mathbf{r}(A)$ we define a predicate $\mathbf{r}(\mathcal{P})$ for every predicate $\mathcal{P}$, where $\mathbf{r}(\mathcal{P})$ has one extra argument place for program terms.

If A is non-computational:
$$\mathbf{r}(A) = \{() \mid A\}$$
If A is non-computational but B is:
$$\mathbf{r}(A \wedge B) = \mathbf{r}(B \wedge A) = \{x \mid A \wedge \mathbf{r}(B)(x)\}$$
$$\mathbf{r}(A \to B) = \{x \mid A \to \mathbf{r}(B)(x)\}$$
In all other cases:
$$\mathbf{r}(\mathcal{P}(\boldsymbol{t})) = \{x \mid \mathbf{r}(\mathcal{P})(x, \boldsymbol{t})\}$$
$$\mathbf{r}(A \wedge B) = \{\langle x, y \rangle \mid \mathbf{r}(A)(x) \wedge \mathbf{r}(B)(y)\}$$
$$\mathbf{r}(A \vee B) = \{\text{inl}(x) \mid \mathbf{r}(A)(x)\} \cup \{\text{inr}(y) \mid \mathbf{r}(B)(y)\}$$
$$\mathbf{r}(A \to B) = \{f \mid \forall x\,(\mathbf{r}(A)(x) \to \mathbf{r}(B)(fx)\}$$
$$\mathbf{r}(\forall y\, A) = \{x \mid \forall y\,(\mathbf{r}(A)(x))\}$$
$$\mathbf{r}(\exists y\, A) = \{x \mid \exists y\,(\mathbf{r}(A)(x))\}$$
If $\mathcal{P}$ is non-computational:

$$\mathbf{r}(\mathcal{P}) = \{((), \boldsymbol{x}) \mid \mathcal{P}(\boldsymbol{x})\}$$

Otherwise:

$$\mathbf{r}(\{\boldsymbol{x} \mid A\}) = \{(y, \boldsymbol{x}) \mid \mathbf{r}(A)(\boldsymbol{x})\}$$
$$\mathbf{r}(X) = \tilde{X}$$
$$\mathbf{r}(\mu X.\mathcal{P}) = \mu \tilde{X}.\mathbf{r}(\mathcal{P})$$
$$\mathbf{r}(\nu X.\mathcal{P}) = \nu \tilde{X}.\mathbf{r}(\mathcal{P})$$

If one uses for operators $\Phi = \lambda X.\mathcal{P}$ the notation $\mathbf{r}(\Phi) := \lambda \tilde{X}.\mathbf{r}(\mathcal{P})$ one can shorten the last two clauses to

$$\mathbf{r}(\mu\Phi) = \mu\mathbf{r}(\Phi)$$
$$\mathbf{r}(\nu\Phi) = \nu\mathbf{r}(\Phi)$$

We call a $\mathcal{L}$-formula a *data formula* if it contains no free predicate variables and every subformula which is the premise of an implication or of the form $\nu\Phi(\boldsymbol{t})$ is non-computational. We also define inductively a unary predicate Data by

$$\text{Data} = \{()\} \cup \text{inl}(\text{Data}) \cup \text{inr}(\text{Data}) \cup \langle\text{Data}, \text{Data}\rangle$$

Of course, this definition is shorthand for

$$\text{Data} := \mu\tilde{X}.\{()\} \cup \{\text{inl}(x) \mid x \in \tilde{X}\} \cup \{\text{inr}(x) \mid x \in \tilde{X}\} \cup \{\langle x, y\rangle \mid x, y \in \tilde{X}\}$$

Lemma 4 (Data formulas). $\mathbf{r}(A) \subseteq \text{Data}$ *for every data formula A.*

Proof. We show more generally: if A is a formula such that every subformula which is the premise of an implication or of the form $\nu\Phi(\boldsymbol{t})$ is non-computational, then

$$\tilde{X}_1 \subseteq \text{Data} \wedge \ldots \wedge \tilde{X}_n \subseteq \text{Data} \rightarrow \mathbf{r}(A) \subseteq \text{Data}$$

where the list $X_1, \ldots, X_n$ comprises all free predicate variables of A (and hence the $\tilde{X}_i$ comprise all free predicate variables of $\mathbf{r}(A)$). The proof is by induction on the structure of A. All cases are straightforward, except induction. In the latter case one shows $\mu\mathbf{r}(\Phi) \subseteq \text{Data}$ by induction. Hence, it suffices to show $\mathbf{r}(\Phi)(\text{Data}) \subseteq \text{Data}$ which follows from $\tilde{X} \subseteq \text{Data} \rightarrow \mathbf{r}(\Phi)(\tilde{X}) \subseteq \text{Data}$ where X (and hence $\tilde{X}$) is a fresh predicate variable. But the latter is an instance of the (structural) induction hypothesis.

Theorem 1 (Soundness). *From a closed derivation of a formula A one can extract a program term M and a derivation of $\mathbf{r}(A)(M)$.*

We prove the Soundness Theorem in the next chapter.

Let us see what we get when we apply realisability to our examples from the Introduction. In the first example, $\mathbf{r}(\mathbb{N})$ is the least relation such that

$$\mathbf{r}(\mathbb{N}) = \{(\text{inl}(()), 0)\} \cup \{(\text{inr}(n), x + 1) \mid \mathbf{r}(\mathbb{N})(n, x)\}$$

Hence, we have for a data d and $x \in \mathbb{R}$ that $d\,\mathbf{r}\,\mathbb{N}(x)$ holds iff x is a natural number and $d = \underline{x} := \mathrm{inr}^x(\mathrm{inl}(()))$, i.e. d is a unary representation of x.

In the second example we first note that the formula $\mathrm{SD}(i)$ is shorthand for the formula $i = 0 \vee i = 1 \vee i = -1$. Hence for suitable data d_i $(i \in \mathrm{SD})$ we have that $\mathbf{r}(\mathrm{C}_0)$ is the largest predicate such that

$$\mathbf{r}(\mathrm{C}_0) = \{(\langle d_i, a \rangle, \mathrm{av}_i(y)) \mid i \in \mathrm{SD},\ y \in \mathbb{I},\ \mathbf{r}(\mathrm{C}_0)(a, y)\}$$

Hence, semantically, $\mathbf{r}(\mathrm{C}_0)(a, y)$ means that $a = a_0, a_1, \ldots$ is an infinite stream of digits $a_i \in \{0, 1, -1\}$ such that $y = \Sigma_{i=0}^{\infty} 2^{-(i+1)} * a_i$.

In the third example we have

$$\mathbf{r}(\mathrm{C}_1) = \nu\tilde{F}\,.\,\mu\tilde{G}\,.\ \ \{(\langle d_i, t \rangle, \mathrm{av}_i \circ f) \mid i \in \mathrm{SD},\ f \in \mathbb{I}^{\mathbb{I}},\ \tilde{F}(t, f)\}\ \cup$$
$$\{(\langle t_0, t_1, t_{-1} \rangle, f) \mid \forall i \in \mathrm{SD}\ \tilde{G}(t_i, g \circ \mathrm{av}_i)\}$$

One sees that a realiser of $\mathrm{C}_1(f)$ is a non-wellfounded tree with two kinds of nodes: "writing nodes" labelled with (a representation of) a signed digit, which means the algorithm writes that digit to the output without reading the input stream, and "reading nodes" where the tree branches into three subtrees meaning that the algorithm reads the first digit of the input stream and continuous with the branch corresponding to the digit read and the tail of the input stream. Due to the inner "$\mu\tilde{G}$" infinitely writing nodes occur on each path through the tree ensuring that in the limit an infinite output stream is produced (see [Ber] for details).

4 Proof of the Soundness Theorem

The main task in proving the Soundness Theorem (Thm. 1) is to define the realisers of induction and coinduction and to prove their correctness.

We define program terms $\mathbf{map}_{X,A}$, $\mathbf{map}_{X,\mathcal{P}}$, $\mathbf{It}_{X,\mathcal{P}}$, and $\mathbf{Coit}_{X,\mathcal{P}}$, where X is a predicate variable, A is formula and $\mathcal{P}$ is a predicate, both strictly positive in X. In Lemma 8 we will show that $\mathbf{map}_{X,\mathcal{P}}$ realises the monotonicity of $\mathcal{P}$ w.r.t. X. Since we haven't yet proven the Soundness Theorem, we cannot extract the map-programs from Lemma 2, but have to code them "by hand". The terms $\mathbf{It}_{X,\mathcal{P}}$ and $\mathbf{Coit}_{X,\mathcal{P}}$ will be used to realise induction and coinduction. In [MP05] the iterators and coiterators are given as constants which expect map-terms as extra arguments, and the property stated in Lemma 8 is an assumption in the Soundness Theorem.

Here, the terms $\mathbf{map}_{X,A}$, $\mathbf{map}_{X,\mathcal{P}}$, $\mathbf{It}_{X,\mathcal{P}}$, and $\mathbf{Coit}_{X,\mathcal{P}}$ are defined by recursion on the structure of A and $\mathcal{P}$. We write $M \circ N$ as an abbreviation for $\lambda x.M(N\,x)$ where x is fresh.

$$\mathbf{map}_{X,A} = \lambda f \lambda x\,.\,x \qquad \text{if } X \text{ is not free in } A, \text{ otherwise}$$
$$\mathbf{map}_{X,\mathcal{P}(t)} = \mathbf{map}_{X,\mathcal{P}}$$
$$\mathbf{map}_{X,A \wedge B} = \lambda f \lambda x\,.\,\langle \mathbf{map}_{X,A} f\,(\pi_1(x)), \mathbf{map}_{X,B} f\,(\pi_2(x)) \rangle$$

$$\mathbf{map}_{X,A\vee B} = \lambda f \lambda x . \operatorname{case} x \operatorname{of} \{\operatorname{inl}(y) \to \mathbf{map}_{X,A} f\, y\,;\, \operatorname{inr}(z) \to \mathbf{map}_{X,B} f\, z\}$$
$$\mathbf{map}_{X,A\to B} = \lambda f \lambda g . \mathbf{map}_{X,B} f \circ g$$

$$\mathbf{map}_{X,\mathcal{P}} = \lambda f \lambda x . x \qquad \text{if } X \text{ is not free in } \mathcal{P}, \text{ otherwise}$$
$$\mathbf{map}_{X,\{\boldsymbol{x}|A\}} = \mathbf{map}_{X,A}$$
$$\mathbf{map}_{X,X} = \lambda f . f$$
$$\mathbf{map}_{X,\mu Y.\mathcal{P}} = \lambda f . \mathbf{It}_{Y,\mathcal{P}}(\mathbf{map}_{X,\mathcal{P}}f)$$
$$\mathbf{map}_{X,\nu Y.\mathcal{P}} = \lambda f . \mathbf{Coit}_{Y,\mathcal{P}}(\mathbf{map}_{X,\mathcal{P}}f)$$

$$\mathbf{It}_{X,\mathcal{P}} = \lambda s . \operatorname{rec} g . s \circ \mathbf{map}_{X,\mathcal{P}}g$$
$$\mathbf{Coit}_{X,\mathcal{P}} = \lambda s . \operatorname{rec} g . \mathbf{map}_{X,\mathcal{P}}g \circ s$$

Lemma 5. *(a)* $\mathbf{It}_{X,\mathcal{P}}s = s \circ \mathbf{map}_{X,\mathcal{P}}(\mathbf{It}_{X,\mathcal{P}}s)$
(b) $\mathbf{Coit}_{X,\mathcal{P}}s = \mathbf{map}_{X,\mathcal{P}}(\mathbf{Coit}_{X,\mathcal{P}}s) \circ s$
(c) $\mathbf{map}_{X,\mu Y.\mathcal{P}}g = \mathbf{map}_{X,\mathcal{P}}g \circ \mathbf{map}_{Y,\mathcal{P}}(\mathbf{map}_{X,\mu Y.\mathcal{P}}g)$
(d) $\mathbf{map}_{X,\nu Y.\mathcal{P}}g = \mathbf{map}_{Y,\mathcal{P}}(\mathbf{map}_{X,\nu Y.\mathcal{P}}g) \circ \mathbf{map}_{X,\mathcal{P}}g$

Proof. (a) and (b) follow immediately from the definitions. For (c) we calculate

$$\mathbf{map}_{X,\mu X.\mathcal{P}}g = \mathbf{It}_{Y,\mathcal{P}}(\mathbf{map}_{X,\mathcal{P}}g)$$
$$= \mathbf{map}_{X,\mathcal{P}}g \circ \mathbf{map}_{Y,\mathcal{P}}(\mathbf{It}_{Y,\mathcal{P}}(\mathbf{map}_{X,\mathcal{P}}g))$$
$$= \mathbf{map}_{X,\mathcal{P}}g \circ \mathbf{map}_{Y,\mathcal{P}}(\mathbf{map}_{X,\mu Y.\mathcal{P}}g)$$

The proof of (d) is similar to the proof of (c).

Lemma 6 (Substitution). *For every operator Φ and predicate $\mathcal{Q}$*

$$\mathbf{r}(\Phi)(\mathbf{r}(\mathcal{Q})) = \mathbf{r}(\Phi(\mathcal{Q}))$$

Proof. Straightforward induction on the (syntactic) size of Φ.

In the next lemmas we consider predicates in the language $\mathbf{r}(\mathcal{L})$ whose first arguments range over predicate terms. The following definitions will be used:

$$\mathcal{P} \circ f := \{(x,\boldsymbol{y}) \mid (f\, x, \boldsymbol{y}) \in \mathcal{P}\}$$
$$f * \mathcal{P} := \{(f\, x, \boldsymbol{y}) \mid (x, \boldsymbol{y}) \in \mathcal{P}\}$$

Clearly, $(\mathcal{P} \circ f) \circ g = \mathcal{P} \circ (f \circ g)$ and $f * (g * \mathcal{P}) = (f * g) * \mathcal{P}$. The rationale for the first of the two definitions is that for predicates $\mathcal{P}$, $\mathcal{Q}$,

$$\mathbf{r}(\mathcal{P} \subseteq \mathcal{Q}) = \{f \mid \mathbf{r}(\mathcal{P}) \subseteq \mathbf{r}(\mathcal{Q}) \circ f\}$$

and the Induction Axiom is an implication between inclusions of predicates. The following easy lemma shows that the two definitions are adjoints. This will allow us to treat induction and coinduction in a similar way.

Lemma 7 (Adjunction). $\mathcal{Q} \subseteq \mathcal{P} \circ f \iff f * \mathcal{Q} \subseteq \mathcal{P}$

Lemma 8 (Map). *Let Φ be an operator in the language $\mathcal{L}$. and X a fresh predicate variable. Then $\mathbf{map}_{X,\Phi(X)}$ realises the monotonicity of Φ, that is*

$$\mathbf{map}_{X,\Phi(X)}\,\mathbf{r}\,(\mathcal{P} \subseteq \mathcal{Q} \to \Phi(\mathcal{P}) \subseteq \Phi(\mathcal{Q}))$$

for all $\mathcal{L}$-predicates $\mathcal{P}$ and $\mathcal{Q}$. By the definition of realisability and the Adjunction Lemma this is equivalent to each of the following two statements about arbitrary $\mathbf{r}(\mathcal{L})$-predicates $\mathcal{P}$ and $\mathcal{Q}$ of appropriate arity and all f:

(a) $\mathcal{P} \subseteq \mathcal{Q} \circ f \to \mathbf{r}(\Phi)(\mathcal{P}) \subseteq \mathbf{r}(\Phi)(\mathcal{Q}) \circ \mathbf{map}_{X,\Phi(X)}f$
(b) $f * \mathcal{P} \subseteq \mathcal{Q} \to \mathbf{map}_{X,\Phi(X)}f * \mathbf{r}(\Phi)(\mathcal{P}) \subseteq \mathbf{r}(\Phi)(\mathcal{Q})$

Furthermore, setting in (a) $\mathcal{P} := \mathcal{Q} \circ f$ and in (b) $\mathcal{Q} := f * \mathcal{P}$ one obtains

(c) $\mathbf{r}(\Phi)(\mathcal{Q} \circ f) \subseteq \mathbf{r}(\Phi)(\mathcal{Q}) \circ \mathbf{map}_{X,\Phi(X)}f$
(d) $\mathbf{map}_{X,\Phi(X)}f * \mathbf{r}(\Phi)(\mathcal{P}) \subseteq \mathbf{r}(\Phi)(f * \mathcal{P})$

Proof. We show a slight generalisation of (a). Let Φ be an operator of $n+1$ arguments, and $X,\boldsymbol{Y}$ fresh predicate variables. Let $\boldsymbol{\mathcal{Q}} = \mathcal{Q}_1,\ldots,\mathcal{Q}_n$ be predicates in the language $\mathbf{r}(\mathcal{L})$. Then for all f, $\mathcal{P}$, $\mathcal{Q}$

$$\mathcal{P} \subseteq \mathcal{Q} \circ f \to \mathbf{r}(\Phi)(\mathcal{P},\boldsymbol{\mathcal{Q}}) \subseteq \mathbf{r}(\Phi)(\mathcal{Q},\boldsymbol{\mathcal{Q}}) \circ \mathbf{map}_{X,\Phi(X)}f$$

The proof is by induction on the structure of $\Phi(X,\boldsymbol{Y})$. In the proof we allow ourselves to switch between (a) and (b) whenever convenient.

Case X does not occur freely in $\Phi(X,\boldsymbol{Y})$. Then $\mathbf{map}_{X,\Phi(X,\boldsymbol{Y})}f$ is the identity. Furthermore, the operator $\mathbf{r}(\Phi)$ does not depend on its first argument. Therefore, the assertion clearly holds.

In the following we assume that X does occur freely in $\Phi(X,\boldsymbol{Y})$.

We only look at the remaining interesting cases, namely those where $\Phi(X,\boldsymbol{Y})$ is X, $\mu Z.\Phi_0(X,\boldsymbol{Y},Z)$ or $\nu Z.\Phi_0(X,\boldsymbol{Y},Z)$.

Case $\Phi(X,\boldsymbol{Y}) = X$. Then $\mathbf{r}(\Phi)(\tilde{X},\tilde{\boldsymbol{Y}}) = \tilde{X}$. Since $\mathbf{map}_{X,X}f = f$, the assertion holds.

Case $\Phi(X,\boldsymbol{Y}) = \mu Z.\Phi_0(X,\boldsymbol{Y},Z)$. Then $\mathbf{r}(\Phi)(\tilde{X},\tilde{\boldsymbol{Y}}) = \mu\tilde{Z}.\mathbf{r}(\Phi_0)(\tilde{X},\tilde{\boldsymbol{Y}},\tilde{Z})$. Assume $\mathcal{P} \subseteq \mathcal{Q} \circ f$. Setting $\mathcal{R} := \mathbf{r}(\Phi)(\mathcal{Q},\boldsymbol{\mathcal{Q}}) = \mu\tilde{Z}.\mathbf{r}(\Phi_0)(\mathcal{Q},\boldsymbol{\mathcal{Q}},\tilde{Z})$, we have to show

$$\mu\tilde{Z}.\mathbf{r}(\Phi_0)(\mathcal{P},\boldsymbol{\mathcal{Q}},\tilde{Z}) \subseteq \mathcal{R} \circ \mathbf{map}_{X,\Phi(X,\boldsymbol{Y})}f$$

We use induction on $\mu\tilde{Z}.\mathbf{r}(\Phi_0)(\mathcal{P},\boldsymbol{\mathcal{Q}},\tilde{Z})$. Hence, we have to show

$$\mathbf{r}(\Phi_0)(\mathcal{P},\boldsymbol{\mathcal{Q}},\mathcal{R} \circ \mathbf{map}_{X,\Phi(X,\boldsymbol{Y})}f) \subseteq \mathcal{R} \circ \mathbf{map}_{X,\Phi(X,\boldsymbol{Y})}f$$

$$\mathbf{r}(\Phi_0)(\mathcal{P},\boldsymbol{\mathcal{Q}},\mathcal{R} \circ \mathbf{map}_{X,\Phi(X,\boldsymbol{Y})}f)$$

$\overset{\text{i.h.(c)}}{\subseteq} \quad \mathbf{r}(\Phi_0)(\mathcal{P},\boldsymbol{\mathcal{Q}},\mathcal{R}) \circ \mathbf{map}_{Z,\Phi_0(X,\boldsymbol{Y},Z)}(\mathbf{map}_{X,\Phi(X,\boldsymbol{Y})}f)$

$\overset{\text{i.h.(a)}}{\subseteq} \quad \mathbf{r}(\Phi_0)(\mathcal{Q},\boldsymbol{\mathcal{Q}},\mathcal{R}) \circ \mathbf{map}_{X,\Phi_0(X,\boldsymbol{Y},Z)}f \circ \mathbf{map}_{Z,\Phi_0(X,\boldsymbol{Y},Z)}(\mathbf{map}_{X,\Phi(X,\boldsymbol{Y})}f)$

$$\overset{\text{Lemma 5 (c)}}{=} \quad \mathbf{r}(\varPhi_0)(\mathcal{Q}, \boldsymbol{\mathcal{Q}}, \mathcal{R}) \circ \mathbf{map}_{X, \varPhi(X, \boldsymbol{Y})} f$$

$$= \quad \mathbf{r}(\varPhi_0)(\mathcal{Q}, \boldsymbol{\mathcal{Q}}, \mu \tilde{Z}.\mathbf{r}(\varPhi_0)(\mathcal{P}, \boldsymbol{\mathcal{Q}}, \tilde{Z})) \circ \mathbf{map}_{X, \varPhi(X, \boldsymbol{Y})} f$$

$$\overset{\text{Fixed Point}}{=} \quad \mu \tilde{Z}.\mathbf{r}(\varPhi_0)(\mathcal{Q}, \boldsymbol{\mathcal{Q}}, \tilde{Z}) \circ \mathbf{map}_{X, \varPhi(X, \boldsymbol{Y})} f$$

$$= \quad \mathcal{R} \circ \mathbf{map}_{X, \varPhi(X, \boldsymbol{Y})} f$$

Case $\varPhi(X, \boldsymbol{Y}) = \nu Z.\varPhi_0(X, \boldsymbol{Y}, Z)$. Then $\mathbf{r}(\varPhi)(\tilde{X}, \tilde{\boldsymbol{Y}}) = \nu \tilde{Z}.\mathbf{r}(\varPhi_0)(\tilde{X}, \tilde{\boldsymbol{Y}}, \tilde{Z})$. Obviously, it is now more convenient to show (b). Assume $f * \mathcal{P} \subseteq \mathcal{Q}$. Setting $\mathcal{R} := \mathbf{r}(\varPhi)(\mathcal{P}, \boldsymbol{\mathcal{Q}}) = \nu \tilde{Z}.\mathbf{r}(\varPhi_0)(\mathcal{P}, \boldsymbol{\mathcal{Q}}, \tilde{Z})$, we have to show

$$\mathbf{map}_{X, \varPhi(X, \boldsymbol{Y})} f * \mathcal{R} \subseteq \nu \tilde{Z}.\mathbf{r}(\varPhi_0)(\mathcal{Q}, \boldsymbol{\mathcal{Q}}, \tilde{Z})$$

We use coinduction on $\nu \tilde{Z}.\mathbf{r}(\varPhi_0)(\mathcal{Q}, \boldsymbol{\mathcal{Q}}, \tilde{Z})$. The proof is exactly dual to the inductive proof above (using the i.h. in the form (d) and (b)).

Proof of the Soundness Theorem (Thm. 1). As usual, one shows by induction on derivations the following more general statement: From a derivation $B_1, \ldots, B_n \vdash A$ one can extract a program term M with free variables among $x_1, \ldots, x_n$ such that $\mathbf{r}(B_1)(x_1), \ldots, \mathbf{r}(B_n)(x_n) \vdash \mathbf{r}(A)(M)$. The only interesting cases are (Co)closure and (Co)induction. Hence, let $\varPhi$ be an operator.

Closure. We have $\mathbf{r}(\varPhi(\mu\varPhi) \subseteq \mu\varPhi) = \{f \mid \mathbf{r}(\varPhi(\mu\varPhi)) \subseteq \mathbf{r}(\mu\varPhi) \circ f\} \overset{\text{Subst. L.}}{=} \{f \mid \mathbf{r}(\varPhi)(\mu\mathbf{r}(\varPhi)) \subseteq \mu\mathbf{r}(\varPhi) \circ f\}$. Hence, we can choose M to be the identity and apply the Closure Axiom for $\mathbf{r}(\varPhi)$.

Coclosure. Similar. Again, the identity is a realiser.

Induction. By the Substitution Lemma, we have

$$\mathbf{r}(\varPhi(\mathcal{Q}) \subseteq \mathcal{Q} \to \mu\varPhi \subseteq \mathcal{Q}) = \{f \mid \forall s\, (\mathbf{r}(\varPhi)(\mathbf{r}(\mathcal{Q})) \subseteq \mathbf{r}(\mathcal{Q}) \circ s \to \mu\mathbf{r}(\varPhi) \subseteq \mathbf{r}(\mathcal{Q}) \circ f s)\}$$

Hence, in order to show that $\mathbf{It}_{\varphi(\alpha)}$ $(=: M)$ realises induction, we assume

$$\mathbf{r}(\varPhi)(\mathbf{r}(\mathcal{Q})) \subseteq \mathbf{r}(\mathcal{Q}) \circ s$$

and show $\mu\mathbf{r}(\varPhi) \subseteq \mathbf{r}(\mathcal{Q}) \circ \mathbf{It}_{\varphi(\alpha)} s$. We use induction on $\mu\mathbf{r}(\varPhi)$, which reduces the problem to showing $\mathbf{r}(\varPhi)(\mathbf{r}(\mathcal{Q}) \circ \mathbf{It}_{\varphi(\alpha)} s) \subseteq \mathbf{r}(\mathcal{Q}) \circ \mathbf{It}_{\varphi(\alpha)} s$.

$$\mathbf{r}(\varPhi)(\mathbf{r}(\mathcal{Q}) \circ \mathbf{It}_{\varphi(\alpha)}) s \overset{\text{Map Lemma (c)}}{\subseteq} \mathbf{r}(\varPhi)(\mathbf{r}(\mathcal{Q})) \circ \mathbf{map}_{\varphi(\alpha)}(\mathbf{It}_{\varphi(\alpha)} s)$$

$$\overset{\text{assumption}}{\subseteq} \mathbf{r}(\mathcal{Q}) \circ s \circ \mathbf{map}_{\varphi(\alpha)}(\mathbf{It}_{\varphi(\alpha)} s)$$

$$\overset{\text{Lemma 5 (a)}}{=} \mathbf{r}(\mathcal{Q}) \circ \mathbf{It}_{\varphi(\alpha)} s$$

Coinduction. Similar, using the Map Lemma (d) and Lemma 5 (b).

5 Semantics of program terms

Now we study a call-by-name operational semantics of program terms which allows us to use the program terms extracted from a formal proof (according

to the Soundness Theorem) as programs that compute useful data. The situation can be illustrated by our first two examples. Suppose we have a proof of $C_0(\pi/4)$. Then the Soundness Theorem yields a program term S and a proof of $S \,\mathbf{r}\, C_0(\pi/4)$. Es explained in Sect. 3, this intuitively means that S denotes an infinite stream of signed binary digits representing $\pi/4$. Hence we cannot expect S to compute an observable (and therefore necessarily finite) data. However, as we will show in Sect. 6, we can conclude from $C_0(\pi/4)$, for example, the formula

$$\exists z \, (\mathbb{Z}(z, 10) \wedge |\pi/4 - z/2^{10}| \leq 1/2^{10}) \tag{1}$$

where $\mathbb{Z}(z, n)$ means that n is a natural number and z is an integer $< n$. The predicate $\mathbb{Z}$ can be defined inductively by

$$\mathbb{Z} = \mu X. \{(0, 0)\} \cup \{(2^n i + z, n + 1) \mid X(z, n) \wedge i \in \mathrm{SD}\}$$

It easy to see that a realiser of $\mathbb{Z}(z, n)$ is a signed binary representation of z. The Soundness Theorem extracts from a proof of (1) a term M denoting an integer z in signed binary (i.e. $M \,\mathbf{r}\, \mathbb{Z}(z)$) such that $|\pi/4 - z/2^{10}| \leq 1/2^{10}$. It remains to be shown that the number z can be computed from the term M. More precisely, we want to compute from M a data term representing the binary representation of z. More generally, we know, by the Soundness Theorem, that from a proof of a formula A we can extract a program term M and a proof of $M \,\mathbf{r}\, A$. Furthermore, if A is a data formula, then we can prove $\mathrm{Data}(M)$, by the Data Lemma. Hence, all that remains to be shown is that whenever $\mathrm{Data}(M)$ is provable, then we can compute from M a data term d such that $M = d$ is provable. In the following we show that this is indeed possible using an operational big-step semantics. As an intermediate step we employ a domain-theoretic *denotational* semantics. The denotational semantics is of independent interest since it directly reflects the intuitive mathematical meaning of program terms. Therefore, we introduce it first. We would also like to stress that our Adequacy Theorem is a general result about an untyped λ-calculus with full recursion which is completely independent of the theory of inductive and coinductive definitions. Hence, if we were to generalise the realisability interpretation to more general kinds of (co)induction (e.g. arbitrary positive or monotone) the Adequacy Theorem needed not be changed.

In the following we mean by a *domain* a *Scott-domain*, i.e. an algebraic, countably based, bounded complete, dcpo [GHK+03]. Note that every domain has a least element $\bot$ w.r.t. the domain ordering $\sqsubseteq$. Let D be the least solution of the domain equation

$$D = \mathbf{1} + D + D + D \times D + [D \to D]$$

where $\mathbf{1}$ is the one-point domain $\{()\}$, and $+, \times, [\cdot \to \cdot]$ denote the usual domain operations, separated sum, cartesian product, and continuous function space. Of course, the domain equation holds only "up to isomorphism", however, we will usually suppress the isomorphism notationally. Hence, every element of D is of exactly one of the following forms: $\bot$, $()$, $\mathrm{inl}(a)$, $\mathrm{inr}(a)$, $\langle a, b \rangle$, $\mathrm{abst}(f)$, where $a, b \in D$ and $f \in [D \to D]$. It follows from standard facts in domain theory

that every program term M defines in a natural way a continuous function $\llbracket M \rrbracket : D^{\mathrm{Var}} \to D$. For example, $\llbracket \lambda x.M \rrbracket \xi = \mathrm{abst}(f)$ where $f(a) = \llbracket M \rrbracket \xi[x \mapsto a]$ and $\llbracket \mathrm{rec}\, x\, . M \rrbracket \xi$ is the least fixed point of f. Furthermore, if $\llbracket M \rrbracket \xi = \mathrm{abst}(f)$, then $\llbracket M\, N \rrbracket \xi = f(\llbracket N \rrbracket \xi)$, otherwise the result is $\bot$.

If Ax is a set of $\vee$-free $\mathcal{L}$-axioms we denote by $\mathbf{r}(\mathrm{Ax})$ the system of $\mathbf{r}(\mathcal{L})$-axioms consisting of the axioms in Ax together with the extra axioms introduced in Sect. 3. If $\mathcal{M}$ is a model of Ax, then we denote by $\mathbf{r}(\mathcal{M})$ the obvious expansion of $\mathcal{M}$ to a model of $\mathbf{r}(\mathrm{Ax})$ using the definition above of the value of a program term. Again, it follows from standard results in domain theory that $\mathbf{r}(\mathcal{M})$ satisfies the axioms for program terms and hence is indeed a model of $\mathbf{r}(\mathrm{Ax})$. Note that in this model the interpretation of the predicate Data defined in Sect. 3 is the least subset $\llbracket \mathrm{Data} \rrbracket$ of D such that

$$\llbracket \mathrm{Data} \rrbracket = \{()\} \cup \mathrm{inl}(\llbracket \mathrm{Data} \rrbracket) \cup \mathrm{inr}(\llbracket \mathrm{Data} \rrbracket) \cup \langle \llbracket \mathrm{Data} \rrbracket, \llbracket \mathrm{Data} \rrbracket \rangle$$

Hence, if $\mathrm{Data}(M)$ is provable, then $\llbracket M \rrbracket \in \llbracket \mathrm{Data} \rrbracket$.

Now we introduce the operational semantics of program terms. A *closure* is a pair (M, η) where M is a program term and η is an *environment*, i.e. a finite mapping from variables to closures, such that all free variables of M are in the domain of η. Note that this is an inductive definition on the meta-level. A *value* is a closure (M, η) where M is an *intro term*, i.e. a term of the form $()$, or $\mathrm{inl}(M_0)$, or $\mathrm{inr}(M_0)$, or $\langle M_1, M_2 \rangle$, or $\lambda x.M_0$. We let $c, c', \dots$ range over closures and $v, v', \dots$ range over values. We inductively define the relation $c \longrightarrow v$ (big-step reduction):

(i) $v \longrightarrow v$

(ii) $\dfrac{\eta(x) \longrightarrow v}{(x, \eta) \longrightarrow v}$

(iii) $\dfrac{(M, \eta) \longrightarrow (\mathrm{inl}(M_0), \eta') \qquad (L, \eta[x \mapsto (M_0, \eta')]) \longrightarrow v}{\mathrm{case}\, M \,\mathrm{of}\{\mathrm{inl}(x) \to L\,;\, \mathrm{inr}(y) \to R\} \longrightarrow v}$

(iv) $\dfrac{(M, \eta) \longrightarrow (\mathrm{inr}(M_0), \eta') \qquad (R, \eta[x \mapsto (M_0, \eta')]) \longrightarrow v}{\mathrm{case}\, M \,\mathrm{of}\{\mathrm{inl}(x) \to L\,;\, \mathrm{inr}(y) \to R\} \longrightarrow v}$

(v) $\dfrac{(M, \eta) \longrightarrow (\langle M_1, M_2 \rangle, \eta') \qquad (M_i, \eta) \longrightarrow v}{\pi_i(M) \longrightarrow v}$

(vi) $\dfrac{(M, \eta) \longrightarrow (\lambda x.M_0, \eta') \qquad (M_0, \eta'[x \mapsto (N, \eta)]) \longrightarrow v}{(M\, N, \eta) \longrightarrow v}$

(vii) $\dfrac{(M, \eta[x \mapsto (\mathrm{rec}\, x\, . M, \eta)]) \longrightarrow v}{(\mathrm{rec}\, x\, . M, \eta) \longrightarrow v}$

Finally, in order to compute data we need a 'print' relation $c \Longrightarrow d$ between closures c and data terms d.

(i) $\dfrac{c \longrightarrow ((), \eta)}{c \Longrightarrow ()}$

(ii) $\dfrac{c \longrightarrow (\mathrm{inl}(M), \eta) \qquad (M, \eta) \Longrightarrow d}{c \Longrightarrow \mathrm{inl}(d)}$

(iii) $\dfrac{c \longrightarrow (\mathrm{inr}(M), \eta) \qquad (M, \eta) \Longrightarrow d}{c \Longrightarrow \mathrm{inr}(d)}$

(iv) $\dfrac{c \longrightarrow (\langle M_1, M_2 \rangle, \eta) \qquad (M_1, \eta) \Longrightarrow d_1 \qquad (M_2, \eta) \Longrightarrow d_2}{c \Longrightarrow \langle d_1, d_2 \rangle}$

Clearly, the inductive definition of $c \Longrightarrow d$ gives rise to an algorithm computing d from c. Since this algorithm corresponds to a call-by-name evaluation of terms one can conclude that whenever $M \Longrightarrow d$, then in a call-by-name language such as Haskell the evaluation of the program corresponding will terminate with a result corresponding to d.

To every closure c we assign a closed term $\bar{c}$ by 'flattening', i.e. removing the structure provided by the nested environments:

$$\overline{(M, \eta)} = M[\overline{\eta(x)}/x \mid x \in \mathrm{dom}(\eta)]$$

Note that this is a recursive definition on the meta-level.

Lemma 9 (Correctness).

(a) If $c \longrightarrow v$, then $\bar{c} = \bar{v}$ is provable.
(b) If $c \Longrightarrow d$, then $\bar{c} = d$ is provable.

Proof. (a) can be proven by straightforward induction on the definition of $c \longrightarrow v$. (b) Follows from (a) and induction on the definition of $c \Longrightarrow d$.

Theorem 2 (Adequacy). *If $[\![M]\!] = d$, then $(M, \emptyset) \Longrightarrow d$.*

The proof of the Adequacy Theorem is obtained by transferring Plotkin's Adequacy Theorem for PCF [Plo77] to the untyped setting by using a a domain-theoretic variant of the reducibility or candidate method [Gir71,Tai75] introduced by Coquand and Spiwack in [CS06]. In loc. cit. and in [Ber08] the technique was applied to prove strong normalisation of λ-calculi with constants and rewrite rules. To carry out the proof, we first exploit the algebraicity of the domain D. Every element of D is the directed supremum of *compact* elements, i.e. elements of D that are generated at some finite stage in the construction of D. Let D_0 be the set of compact elements of D. There is a rank function $\mathbf{rk}(\cdot) : D_0 \to \mathbb{N}$ with the following properties:

(rk1) The images of the injections $\mathrm{inl}(\cdot), \mathrm{inr}(\cdot)$, and the pairing function $\langle \cdot, \cdot \rangle$ are compact iff their arguments are. Furthermore, injections and pairing increase rank.

(rk2) If $\mathrm{abst}(f)$ is compact, then for every $a \in D$, $f(a)$ is compact with $\mathbf{rk}(f(a)) < \mathbf{rk}(\mathrm{abst}(f))$, and there exists a compact $a_0 \sqsubseteq a$ with $\mathbf{rk}(a_0) < \mathbf{rk}(a)$ and $f(a_0) = f(a)$.

These properties allow us to define for every compact a a set $\mathbf{Cl}(a)$ of closures, by recursion on $\mathbf{rk}(a)$:

$$\mathbf{Cl}(\bot) = \text{ the set of all closures}$$
$$\mathbf{Cl}(()) = \{c \mid \exists \eta \, (c \longrightarrow ((), \eta))\}$$
$$\mathbf{Cl}(\mathrm{inl}(a)) = \{c \mid \exists (M, \eta) \in \mathbf{Cl}(a) \, (c \longrightarrow (\mathrm{inl}(M), \eta))\}$$
$$\mathbf{Cl}(\mathrm{inr}(a)) = \{c \mid \exists (M, \eta) \in \mathbf{Cl}(a) \, (c \longrightarrow (\mathrm{inr}(M), \eta))\}$$
$$\mathbf{Cl}(\langle a_1, a_2 \rangle) = \{c \mid \exists M_1, M_2, \eta \, ((M_1, \eta) \in \mathbf{Cl}(a_1) \wedge (M_1, \eta) \in \mathbf{Cl}(a_1) \wedge$$
$$c \longrightarrow (\langle M_1, M_2 \rangle, \eta))\}$$
$$\mathbf{Cl}(f) = \{c \mid \exists x, M, \eta \, (c \longrightarrow (\lambda x.M, \eta) \wedge \forall a \in D_0 \, (\mathbf{rk}(a) < \mathbf{rk}(\mathrm{abst}(f))$$
$$\rightarrow \forall c' \in \mathbf{Cl}(a) \, (M, \eta[x \mapsto c']) \in \mathbf{Cl}(f(a))))\}$$

Note that the sets $\mathbf{Cl}(a)$ are defined in analogy with the reducibility or computability predicates mentioned above.

Lemma 10. *If a, b are compact with $a \sqsubseteq b$, then $\mathbf{Cl}(a) \supseteq \mathbf{Cl}(b)$.*

Proof. Induction on the maximum of $\mathbf{rk}(a)$ and $\mathbf{rk}(b)$. The only interesting case is $\mathrm{abst}(f) \sqsubseteq \mathrm{abst}(g)$. Then $f \sqsubseteq g$ (pointwise). Let $c \in \mathbf{Cl}(\mathrm{abst}(g))$. Then $c \longrightarrow (\lambda x.M, \eta)$, and for all compact b with $\mathbf{rk}(b) < \mathbf{rk}(\mathrm{abst}(g))$ and all $c' \in \mathbf{Cl}(b)$ we have $(M, \eta[x \mapsto c']) \in \mathbf{Cl}(g(b))$. We show $c \in \mathbf{Cl}(\mathrm{abst}(f))$ using the same witness $(\lambda x.M, \eta)$. Let a be compact with $\mathbf{rk}(a) < \mathbf{rk}(\mathrm{abst}(f))$ and let $c' \in \mathbf{Cl}(a)$. By (rk2), there exists a compact $b \sqsubseteq a$ with $\mathbf{rk}(b) < \mathbf{rk}(\mathrm{abst}(g))$ and $g(b) = g(a)$. By induction hypothesis, $\mathbf{Cl}(b) \supseteq \mathbf{Cl}(a)$, hence $c' \in \mathbf{Cl}(b)$. It follows $(M, \eta[x \mapsto c']) \in \mathbf{Cl}(g(b)) = \mathbf{Cl}(g(a))$.

Lemma 11. $c \in \mathbf{Cl}(a)$ *iff there exists a value v with $c \longrightarrow v$ and $v \in \mathbf{Cl}(a)$.*

Proof. This can be seen by a trivial induction in $\mathbf{rk}(a)$ using the fact that for values v, v' we have $v \longrightarrow v'$ iff $v = v'$.

In the following we write $\eta \in \mathbf{Cl}(\xi)$ if for all $x \in \mathrm{dom}(\eta)$, $\xi(x)$ is compact and $\eta(x) \in \mathbf{Cl}(\xi(x))$.

Lemma 12. *If $c \in \mathbf{Cl}(d)$, where d is a data, then $c \Longrightarrow d$.*

Proof. Straightforward induction on d.

Lemma 13 (Approximation). *If $\eta \in \mathbf{Cl}(\xi)$ and a is compact with $a \sqsubseteq [\![M]\!]\xi$, then $(M, \eta) \in \mathbf{Cl}(a)$.*

Proof. Let $[\![M]\!]^n \xi$ denote the n-th stage in the definition of $[\![M]\!]\xi$. Hence, $[\![M]\!]^0 \xi = \bot$ and e.g. $[\![\lambda x.M]\!]^{n+1} \xi(a) = [\![M]\!]^n \xi[\mapsto a]$, e.t.c. Since the $[\![M]\!]^n \xi$ form an increasing chain in D with $[\![M]\!]\xi$ as its supremum, it follows that if a is compact and $a \sqsubseteq [\![M]\!]\xi$, then $a \sqsubseteq [\![M]\!]^n \xi$ for some n. Hence, it is enough to show:

If $\eta \in \mathbf{Cl}(\xi)$ and a is compact with $a \sqsubseteq [\![M]\!]^n \xi$, then $(M, \eta) \in \mathbf{Cl}(a)$.

We prove the assertion by induction on $n \in \mathbb{N}$. The induction base, $n = 0$, is easy, since $[\![M]\!]^0\xi = \bot$ and therefore $a = \bot$, and $\mathbf{Cl}(\bot)$ is the set of all closures.

In the induction step, $n + 1$, we do a case analysis on the shape of M. We may assume $a \neq \bot$, since otherwise the assertion is trivial.

Case x. By assumption, $a \sqsubseteq [\![x]\!]^{n+1}\xi = \xi(x)$ and $\eta(x) \in \mathbf{Cl}(\xi(x))$. By Lemma 10, $\eta(x) \in \mathbf{Cl}(a)$. By Lemma 11, there exists a value v with $\eta(x) \longrightarrow v$ and $v \in \mathbf{Cl}(a)$. It follows $(x, \eta) \longrightarrow v$ and therefore $(x, \eta) \in \mathbf{Cl}(a)$, again by Lemma 11.

Case (). By assumption, $a \sqsubseteq [\![()]\!]^{n+1}\xi = ()$. Hence $a = ()$ (since $a \neq \bot$). Clearly, $((), \eta) \in \mathbf{Cl}(())$.

Case $\mathrm{inl}(M)$. By assumption, $a \sqsubseteq [\![\mathrm{inl}(M)]\!]^{n+1}\xi = \mathrm{inl}([\![M]\!]^n\xi)$. Hence $a = \mathrm{inl}(a_0)$ with $a_0 \sqsubseteq [\![M]\!]^n\xi =$. By induction hypothesis, $(M\eta) \in \mathbf{Cl}(a_0)$. Since $(\mathrm{inl}(M), \eta) \longrightarrow (\mathrm{inl}(M), \eta)$ $((\mathrm{inl}(M), \eta)$ is a value), it follows $(\mathrm{inl}(M), \eta) \in \mathbf{Cl}(\mathrm{inl}(a_0))$.

Cases $\mathrm{inr}(M), \langle M_1, M_1 \rangle$. Similar.

Case $\lambda x.M$. By assumption, $a \sqsubseteq [\![\lambda x.M]\!]^{n+1}\xi = \mathrm{abst}(g)$ where $g(b) = \mathrm{inl}([\![M]\!]^n\xi[x \mapsto b])$. Hence, $a = \mathrm{abst}(f)$ with $f \sqsubseteq g$. By induction hypothesis, $(M, \eta[x \mapsto c]) \in \mathbf{Cl}(f(b))$, for all compact b and all $c \in \mathbf{Cl}(b)$. Since $(\lambda x.M, \eta) \longrightarrow (\lambda x.M, \eta)$, it follows $(\lambda x.M, \eta) \in \mathbf{Cl}(\mathrm{abst}(f))$.

Case $\mathrm{case}\, M \,\mathrm{of}\{\mathrm{inl}(x) \to L\,;\, \mathrm{inr}(y) \to R\}$. By assumption we have $a \sqsubseteq [\![\mathrm{case}\, M \,\mathrm{of}\{\mathrm{inl}(x) \to L\,;\, \mathrm{inr}(y) \to R\}]\!]^{n+1}\xi$. Since $a \neq \bot$ we have, w.l.o.g. $[\![M]\!]^n\xi = \mathrm{inl}(b)$ and $a \sqsubseteq [\![L]\!]^n\xi[x \mapsto b]$. Since a is compact and the function mapping b to $[\![L]\!]^n\xi[x \mapsto b]$ is continuous it follows that $a \sqsubseteq [\![L]\!]^n\xi[x \mapsto b_0]$ for some compact $b_0 \sqsubseteq b$. By induction hypothesis, $(M, \eta) \in \mathbf{Cl}(\mathrm{inl}(b_0))$. Hence, $(M, \eta) \longrightarrow (\mathrm{inl}(M_0), \eta_0)$ with $(M_0, \eta_0) \in \mathbf{Cl}(b_0)$. Again, by induction hypothesis, $(L, \eta[x \mapsto (M_0, \eta_0)]) \in \mathbf{Cl}(a)$. By Lemma 11, $(L, \eta[x \mapsto (M_0, \eta_0)]) \longrightarrow v$ for some value $v \in \mathbf{Cl}(a)$. It follows $(\mathrm{case}\, M \,\mathrm{of}\{\mathrm{inl}(x) \to L\,;\, \mathrm{inr}(y) \to R\}, \eta) \longrightarrow v$ and consequently $(\mathrm{case}\, M \,\mathrm{of}\{\mathrm{inl}(x) \to L\,;\, \mathrm{inr}(y) \to R\}, \eta) \in \mathbf{Cl}(a)$, again by Lemma 11.

Cases $\pi_i(M)$. Similar.

Case $M\,N$. By assumption, $a \sqsubseteq [\![M\,N]\!]^{n+1}\xi$. Since $a \neq \bot$ we have, $[\![M]\!]^n\xi = \mathrm{abst}(f)$ and $a \sqsubseteq f([\![N]\!]^n\xi)$. Since function application is continuous, there are a compact $f_0 \sqsubseteq f$ and a compact $b \sqsubseteq [\![N]\!]^n\xi$ with $a \sqsubseteq f_0(b)$. By (2) above, we may assume $\mathbf{rk}(b) < \mathbf{rk}(\mathrm{abst}(f_0))$. By induction hypothesis, $(M, \xi) \in \mathbf{Cl}(\mathrm{abst}(f_0))$ and $(N, \eta) \in \mathbf{Cl}(b)$. Therefore, $M \longrightarrow (\lambda x.M_0, \eta_0)$ such that $(M_0, \eta_0[x \mapsto c]) \in \mathbf{Cl}(f_0(b_0))$ for all compact b_0 with $\mathbf{rk}(b_0) < \mathbf{rk}(\mathrm{abst}(f_0))$ and all $c \in \mathbf{Cl}(b_0)$. Applying this to $b_0 := b$ and $c := (N, \eta)$ we obtain $(M_0, \eta_0[x \mapsto (N, \eta)]) \in \mathbf{Cl}(f_0(b))$. By Lemma 11, $(M_0, \eta_0[x \mapsto (N, \eta)]) \longrightarrow v$ for some $v \in \mathbf{Cl}(f_0(b))$. It follows $(M\,N, \eta) \longrightarrow v$ and hence $(M\,N, \eta) \in \mathbf{Cl}(f_0(b)) \subseteq \mathbf{Cl}(a)$, by Lemma 11 and Lemma 10.

Case $\mathrm{rec}\, x\,.\, M$. By assumption, we have $a \sqsubseteq [\![\mathrm{rec}\, x\,.\, M]\!]^{n+1}\xi = [\![M]\!]^n\xi[x \mapsto [\![\mathrm{rec}\, x\,.\, M]\!]^n\xi]$. By a similar continuity argument as earlier in the proof, there exists a compact $b \sqsubseteq [\![\mathrm{rec}\, x\,.\, M]\!]^n\xi$ such that $a \sqsubseteq [\![M]\!]^n\xi[x \mapsto b]$. By induction hypothesis, $(\mathrm{rec}\, x\,.\, M, \eta) \in \mathbf{Cl}(b)$ and $(M, \eta[x \mapsto (\mathrm{rec}\, x\,.\, M, \eta)]) \in \mathbf{Cl}(a)$. By

Lemma 11, $(M, \eta[x \mapsto (\text{rec } x \,.\, M, \eta)]) \longrightarrow v$ for some value $v \in \mathbf{Cl}(a)$, therefore $(\text{rec } x \,.\, M, \eta) \longrightarrow v$, and finally, $(\text{rec } x \,.\, M, \eta) \in \mathbf{Cl}(a)$.

Proof of the Adequacy Theorem (Thm. 2). Assume $[\![M]\!] = d$ for some data d. Since d is compact, it follows, by the Approximation Lemma, $(M, \emptyset) \in \mathbf{Cl}(d)$. Hence $(M, \emptyset) \Longrightarrow d$, by Lemma 12.

6 Program extraction

To conclude, we summarise our results and continue the examples.

Theorem 3 (Program extraction). *From a proof of a data formula A one can extract a program term M with the property that $(M, \emptyset) \Longrightarrow d$ for some data d provably realising A, i.e. $\mathbf{r}(A)(d)$ is provable.*

Proof. By the Soundness Theorem, we obtain from a proof of A a program term M and a proof of $\mathbf{r}(A)(M)$. By Lemma 4, $\text{Data}(M)$ is provable and therefore true in D, i.e. $[\![M]\!] = d$ for some data d. By the Adequacy Theorem, $(M, \emptyset) \Longrightarrow d$, and by Lemma 9, $M = d$ is provable. It follows that $\mathbf{r}(A)(d)$ is provable.

Let us resume our example from Sect. 5.

Lemma 14 (Printing digits).

$$\forall n \, (\mathbb{N}(n) \to \forall x \, (\mathrm{C}_0(x) \to \exists z \, (\mathbb{Z}(z, n) \land |x - \frac{z}{2^n}| \leq \frac{1}{2^n})))$$

Proof. Induction on $\mathbb{N}(n)$. Set $\mathcal{P} := \{n \mid \forall x \, (\mathrm{C}_0(x) \to \exists z \, (\mathbb{Z}(z, n) \land |x - \frac{z}{2^n}| \leq \frac{1}{2^n}))\}$. We have to show (1) $\mathcal{P}(0)$, (2) $\forall n \, (\mathcal{P}(n) \to \mathcal{P}(n+1))$. For (1), we can take $z := 0$, since $\mathrm{C}_0(x)$ implies $|x| \leq 1$. For (2), assume $\mathcal{P}(n)$ (i.h.) and $\mathrm{C}_0(x)$. Let $i \in \mathrm{SD}$ such that $x = \mathrm{av}_i(y)$ for some y with $\mathrm{C}_0(y)$. By i.h. there exists z such that $\mathbb{Z}(z, n)$ and $|y - \frac{z}{2^n}| \leq \frac{1}{2^n}$. It follows $\mathbb{Z}(2^n i + z, n+1)$ and

$$|x - \frac{2^n i + z}{2^{n+1}}| = \frac{1}{2}|y - \frac{z}{2^n}| \leq \frac{1}{2^{n+1}}$$

The program extracted from this proof take as inputs a (unary) natural number n and a signed digit stream a representing some real number in $\mathbb{I}$, and computes an signed binary representation of an integer $z < 2^n$ such that $|x - z/2^n| \leq 1/2^n$. In fact the digits of that representation will be exactly the first n elements of the stream a. Hence, the extracted program is essentially Haskell's function `take` that computes the n first elements of a stream.

7 Conclusion and further work

In this paper we laid the logical and semantical foundations for the extraction of programs from proofs involving inductive and coinductive definitions. The main results where the *Soundness Theorem* for a realisability interpretation stating

that the extracted program provably realises the proven formula, and the *Adequacy Theorem* stating that for *data formulas* the realisers can be computed into canonical form via a call-by-name operational semantics. The proof of the Adequacy Theorem used a domain-theoretic method due to Coquand and Spiwack.

We restricted ourselves to simple examples illustrating the method. More substantial applications are described in [Ber]. Strictly speaking our results do not apply to loc. cit. because there realisers are typed (with Haskell or ML style polymorphic types) while our realisers are untyped. We plan to recast our results with typed realisers, which will probably technically more complicated, but will have the advantage that the category-theoretic justification of induction and coinduction can be used to "derive" the realisability interpretation. Moreover, this will allow for a direct interpretation of realisers as programs in a call-by-name typed programming language such as Haskell.

A major piece of work that remains to be done is to implement the realisability interpretation in an interactive theorem prover and carry out case studies. We expect this to tie in nicely with recent work on implementations of inductive and and coinductive definitions and proofs [CDG06,Ber07]), exact real arithmetic [MRE07,GNSW07,EH02,Sch08], realisability [BS07a], and functional interpretation [HO08].

References

[AMU05]   A. Abel, R. Matthes, and T. Uustalu. Iteration and coiteration schemes for higher-order and nested datatypes. *Theor. Comput. Sci*, 333:3–66, 2005.

[Ber]   U. Berger. From coinductive proofs to exact real arithmetic. Submitted. Draft available from http://www.cs.swan.ac.uk/∼csulrich/.

[Ber07]   Y. Bertot. Affine functions and series with co-inductive real numbers. *Mathematical Structures in Computer Science*, 17:37–63, 2007.

[Ber08]   U. Berger. A domain model characterising strong normalisation. *Annals of Pure and Applied Logic*, 1841:00–00, 2008.

[BFPS81]   W. Buchholz, F. Feferman, W. Pohlers, and W. Sieg. *Iterated Inductive Definitions and Subsystems of Analysis: Recent Proof–Theoretical Studies*, volume 897 of *Lecture Notes in Mathematics*. Springer, Berlin, 1981.

[BS07a]   A. Bauer and A.C. Stone. RZ: A tool for bringing constructive and computable mathematics closer to programming practice. In S.B. Cooper, B. Löwe, and A. Sorbi, editors, *CiE 2007: Computation and Logic in the Real World*, volume 4497 of *LNCS*, pages 28–42, 2007.

[BS07b]   J. Bradfield and C. Stirling. Modal mu-calculi. In P. Blackburn, J. van Benthem, and F. Wolter, editors, *Handbook of Modal Logic*, volume 3 of *Studies in Logic and Practical Reasoning*, pages 721–756. Elsevier, 2007.

[CDG06]   A. Ciaffaglione and P. Di Gianantonio. A certified, corecursive implementation of exact real numbers. *Theor. Comput. Sci*, 351:39–51, 2006.

[CS06]   T. Coquand and A. Spiwack. A proof of strong normalisation using domain theory. In *Proceedings of the 21st Annual IEEE Symposium on Logic in Computer Science (LICS'06)*, pages 307–316. IEEE Computer Society Press, 2006.

[EH02]   A. Edalat and R. Heckmann. Computing with real numbers: I. the lft
           approach to real number computation; ii. a domain framework for com-
           putational geometry. In G. Barthe, P. Dybjer, L. Pinto, and J. Saraiva,
           editors, *Applied Semantics - Lecture Notes from the International Summer
           School, Caminha, Portugal*, pages 193–267. Springer, 2002.

[GHK$^+$03] G. Gierz, K.H. Hofmann, K. Keimel, J.D. Lawson, M. Mislove, and D.S.
           Scott. *Continuous Lattices and Domains*, volume 93 of *Encyclopedia of
           Mathematics and its Applications*. Cambridge University Press, 2003.

[GHP06]   N. Ghani, P. Hancock, and D. Pattinson. Continuous functions on final
           coalgebras. 164, 2006.

[Gir71]   J-Y. Girard. Une extension de l'interprétation de Gödel à l'analyse, et
           son application à l'élimination des coupures dans l'analyse et la théorie des
           types. In J.E. Fenstad, editor, *Proceedings of the Second Scandinavian Logic
           Symposium*, pages 63–92. North–Holland, 1971.

[GNSW07]   H. Geuvers, M. Niqui, B. Spitters, and F. Wiedijk. Constructive analysis,
           types and exact real numbers (overview article). *Mathematical Structures
           in Computer Science*, 17(1):3–36, 2007.

[HO08]   M. D. Hernest and P. Oliva. Hybrid functional interpretations. In A. Beck-
           mann, C. Dimitracopoulos, and B. Löwe, editors, *CiE 2008: Logic and The-
           ory of Algorithms*, volume 5028 of *LNCS*, pages 251–260. Springer, 2008.

[MÖ3]   M. Möllerfeld. *Generalized inductive definitions*. PhD thesis, Westfälische
           Wilhelms-Universität Münster, 2003.

[Mat01]   R. Matthes. Monotone inductive and coinductive constructors of rank 2.
           In L Fribourg, editor, *Computer Science Logic (Proceedings of the Fifteenth
           CSL Conference)*, number 2142 in LNCS, pages 600–615. Springer, 2001.

[Men91]   N.P. Mendler. Inductive types and type constraints in the second-order
           lambda calculus. *Ann. Pure Appl. Logic*, 51:159–172, 1991.

[MP05]   F. Miranda-Perea. Realizability for monotone clausular (co)inductive def-
           initions. *Electronic Notes in Theoretical Computer Science*, 123:179–193,
           2005.

[MRE07]   J. R. Marcial-Romero and M. H. Escardo. Semantics of a sequential lan-
           guage for exact real-number computation. *Theor. Comput. Sci*, 379(1-
           2):120–141, 2007.

[Plo77]   G.D. Plotkin. LCF considered as a programming language. *Theor. Com-
           put. Sci*, 5:223–255, 1977.

[RT09]   D. Ratiu and T. Trifonov. Exploring the computational content of the infi-
           nite pigeonhole principle. To appear in Journal of Logic and Computation,
           2009.

[Sch08]   H. Schwichtenberg. Realizability interpretation of proofs in constructive
           analysis. To appear in ToCS, 2008.

[Tai75]   W.W. Tait. A realizability interpretation of the theory of species. In
           R. Parikh, editor, *Logic Colloquium Boston 1971/72*, volume 453 of *Lec-
           ture Notes in Mathematics*, pages 240–251. Springer, 1975.

[Tat98]   M. Tatsuta. Realizability of monotone coinductive definitions and its appli-
           cation to program synthesis. In R. Parikh, editor, *Mathematics of Program
           Construction*, volume 1422 of *Lecture Notes in Mathematics*, pages 338–364.
           Springer, 1998.

[Tup04]   S. Tupailo. On the intuitionistic strength of monotone inductive definitions.
           Jour. Symb. Logic, 69(3):790–798, 2004.