Journal of Universal Computer Science, vol. 16, no. 18 (2010), 2535-2555
submitted: 14/12/09, accepted: 28/4/10, appeared: 28/9/10 © J.UCS

Realisability for Induction and Coinduction with Applications to Constructive Analysis

Ulrich Berger
(Swansea University, United Kingdom
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.
Key Words: realisability, program extraction, coinduction, constructive analysis
Category: F.3, F.3.1, F.3.2

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 on predicates. The main technical results are the *Soundness Theorem* for this interpretation and the *Computational Adequacy Theorem* for the realisers with respect to a call-by-name operational semantics and a domain-theoretic denotational semantics. Both together imply the *Program Extraction Theorem* stating that from a constructive proof one can extract a program that is *provably correct* and *terminating*.

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* subset of $\mathbb{R}$ 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 on $\mathcal{P}(\mathbb{R})$ 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, we set $\mathbb{I} := [-1, 1] = \{x \mid -1 \leq x \leq 1\} \subseteq \mathbb{R}$, $\mathrm{SD} := \{0, 1, -1\}$ (signed digits), and $\mathrm{av}_i(x) := (x + i)/2$. We 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, classically, $\mathrm{C}_0 = \mathbb{I}$, hence the coinductive definition seems to be unnecessary. However, the point is that in order to prove $\mathrm{C}_0(x)$ for $x \in \mathbb{I}$ constructively, one needs the extra assumption that there is a rational Cauchy sequence converging to x. The (coinductive) proof of $\mathrm{C}_0(x)$ contains a (coiterative) *program* transforming the Cauchy sequence into a signed digit representation of x.

Our third example extends the previous to unary functions. We 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

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

One can show that C_1 coincides with the set of functions in $\mathbb{I}^{\mathbb{I}}$ that are (constructively) uniformly continuous. Moreover, a constructive proof of $\mathrm{C}_1(f)$ contains a program implementing f as a non-wellfounded tree which acts as a (signed digit) stream transformer. The trees generated in this way are similar to the structures studied by Ghani, Hancock and Pattinson [Ghani et al. 2006]. The interpretation of these trees as stream transformers 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 [Berger 2009]. The algorithmic idea embodied in the definition of the predicate C_1 has been used before in [Edalat and Heckmann 2002] and elsewhere to develop exact real number algorithms based on the signed digit and the more general linear fractional transformation representation. One way to look at our paper is that we use inductive/coinductive definitions to give an elegant formalisation of the work in loc. cit. and use program extraction to get correctness proofs for free.

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" fixed points is known as the *μ-calculus* [Bradfield and Stirling 2007]. Möllerfeld [Möllerfeld 2003] analysed 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^1_2-comprehension. Tupailo [Tupailo 2004] 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* [Buchholz et al. 1981].

The realisability interpretation we are going to study is related to interpretations given in [Tatsuta 1998] and [Miranda-Perea 2005]. We 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 is obtained by a general Computational Adequacy Theorem relating the operational with a (domain-theoretic) denotational semantics. In contrast to this, Miranda-Perea extracts typed terms and uses the more general "Mendler-style" (co)inductive definitions [Mendler 1991] which extract strongly normalising terms in extensions of the second-order polymorphic λ-calculus or stronger systems [Matthes 2001, Abel and Matthes 2005]. 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 (as Miranda-Perea does): For universally quantified formulas realisability, $M\,\mathbf{r}\,\forall x\,A(x)$, is defined as $\forall x\,(M\,\mathbf{r}\,A(x))$ (instead of $\forall x\,(M\,x\,\mathbf{r}\,A(x)))$ and $M\,\mathbf{r}\,\exists x\,A(x)$ is defined as $\exists x\,(M\,\mathbf{r}\,A(x))$ (instead of $\pi_2(M)\,\mathbf{r}\,A(\pi_1(M)))$. In general, the object language and the language of realisers are kept strictly separate. This means in particular that a realiser neither depends on variables of the object language nor produces output in that language. Realisers are extracted exclusively from the "propositional skeleton" of a proof ignoring the first-order part. The latter 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 in [Schwichtenberg 2009], [Hernest and Oliva 2008] and [Ratiu and Trifonov 2009], which allow for a fine control of the amount of computational information extracted from proofs. A related method of importing computational information in a controlled way is the realisability interpretation via assemblies in [Bauer and Blanck 2009] where realisers are drawn from an arbitrary partial combinatory algebra (PCA). In constrast, we are working with a fixed PCA of λ-terms with constructors (see Sect. 3).

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 $s = t$, $\mathcal{P}(\mathbf{t})$ 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 $\lambda\mathbf{x}.A$ (sometimes also written $\{\mathbf{x} \mid A\}$) where A is a formula and $\mathbf{x}$ 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, $\mathcal{P}(\mathbf{t})$, of a predicate $\mathcal{P}$ to a list of terms $\mathbf{t}$ is a primitive syntactic construct, except when $\mathcal{P}$ is a comprehension term, $\mathcal{P} = \{\mathbf{x} \mid A\}$, in which case $\mathcal{P}(\mathbf{t})$ stands for $A[\mathbf{t}/\mathbf{x}]$.

It will sometimes be convenient to write $\mathbf{x} \in \mathcal{P}$ instead of $\mathcal{P}(\mathbf{x})$ and also $\mathcal{P} \subseteq \mathcal{Q}$ for $\forall\mathbf{x}\,(\mathcal{P}(\mathbf{x}) \to \mathcal{Q}(\mathbf{x}))$ and $\mathcal{P} \cap \mathcal{Q}$ for $\{\mathbf{x} \mid \mathcal{P}(\mathbf{x}) \wedge \mathcal{Q}(\mathbf{x})\}$, e.t.c. We also write $\{t \mid A\}$ as an abbreviation for $\{x \mid \exists\mathbf{y}\,(x = t \wedge A)\}$ where x is a fresh variable and $\mathbf{y} = \mathrm{FV}(t) \cap \mathrm{FV}(A)$. 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 ones 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

Closure	$\Phi(\mu\Phi) \subseteq \mu\Phi$	Induction	$\Phi(\mathcal{Q}) \subseteq \mathcal{Q} \to \mu\Phi \subseteq \mathcal{Q}$
Coclosure	$\nu\Phi \subseteq \Phi(\nu\Phi)$	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 (classically) in the intended model. For instance, our running examples take place in the classical theory of real closed fields. Since the axioms must be non-computational, we have to express disjunctive properties, such as linearity of the order, negatively, e.g. $\forall x, y\,(x \not< y \wedge y \not< x \to x = y)$. However, it is easy to see that for the natural numbers $\mathbb{N}$, as defined in the introduction, one can prove constructive linearity, $\forall x, y \in \mathbb{N}\,(x < y \vee y < x \vee x = y)$, and hence decidability of equality, $\forall x, y \in \mathbb{N}\,(x = y \vee x \neq y)$. We write $\Gamma \vdash A$ if A is derivable (intuitionistically) from assumptions in Γ in this system. If A is derivable without assumptions we write $\vdash A$, or 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 $\bot \to A$ for every formula A. By induction on derivations one easily proves:

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

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

The following lemma, which is well-known and easy to prove, can be viewed as an instance of Lambek's Lemma [Lambek 1968].

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

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

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

3 Realisability

The realisers of formulas are terms of an untyped λ-calculus with pairing, injections and recursion *Program-terms*, $M, N, K, L, R \dots$ (*terms* for short) are variables $x, y, z, \dots$, 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 $\mathrm{inl}(\cdot)$, $\mathrm{inr}(\cdot)$. We call these terms *data* and denote them by $d, e, \dots$. Roughly speaking, data stand for themselves and will in any reasonable denotational semantics coincide with their value. In Section 5 we 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

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

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 $X_1, \dots, X_n$, then the predicate $\mathbf{r}(A)$ has the free predicate variables $\tilde{X_1}, \dots, \tilde{X_n}$. 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.

$$\begin{aligned}
\mathbf{r}(A) &= \{() \mid A\} \quad \text{if } A \text{ is non-computational}\\
&\text{Otherwise}\\
\mathbf{r}(A \wedge B) = \mathbf{r}(B \wedge A) &= \{x \mid A \wedge \mathbf{r}(B)(x)\} \quad \text{if } A \text{ is non-computational}\\
\mathbf{r}(A \to B) &= \{x \mid A \to \mathbf{r}(B)(x)\} \quad \text{if } A \text{ is non-computational}\\
&\text{Otherwise}\\
\mathbf{r}(\mathcal{P}(\mathbf{t})) &= \{x \mid \mathbf{r}(\mathcal{P})(x, \mathbf{t})\}\\
\mathbf{r}(A \wedge B) &= \{\langle x, y\rangle \mid \mathbf{r}(A)(x) \wedge \mathbf{r}(B)(y)\}\\
\mathbf{r}(A \vee B) &= \{\mathrm{inl}(x) \mid \mathbf{r}(A)(x)\} \cup \{\mathrm{inr}(y) \mid \mathbf{r}(B)(y)\}\\
\mathbf{r}(A \to B) &= \{f \mid \forall x\,(\mathbf{r}(A)(x) \to \mathbf{r}(B)(fx)\}\\
\mathbf{r}(\forall y\, A) &= \{x \mid \forall y\,(\mathbf{r}(A)(x))\}\\
\mathbf{r}(\exists y\, A) &= \{x \mid \exists y\,(\mathbf{r}(A)(x))\}
\end{aligned}$$

$$\begin{aligned}
\mathbf{r}(\mathcal{P}) &= \{((), \mathbf{x}) \mid \mathcal{P}(\mathbf{x})\} \quad \text{if } \mathcal{P} \text{ is non-computational}\\
&\text{Otherwise}\\
\mathbf{r}(\{\mathbf{x} \mid A\}) &= \{(y, \mathbf{x}) \mid \mathbf{r}(A)(y)\}\\
\mathbf{r}(X) &= \tilde{X}\\
\mathbf{r}(\mu X.\mathcal{P}) &= \mu\tilde{X}.\mathbf{r}(\mathcal{P})\\
\mathbf{r}(\nu X.\mathcal{P}) &= \nu\tilde{X}.\mathbf{r}(\mathcal{P})
\end{aligned}$$

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 an $\mathcal{L}$-formula a *parametric data formula* if every subformula of the form $A \to B$ or $\nu\Phi(\mathbf{t})$ is non-computational. A *data formula* is a parametric data formula without free predicate variables. We also define inductively a unary predicate Data by

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

Of course, this definition is shorthand for

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

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

Proof. We show that for every parametric data formula A

$$\mathbf{r}(A)' \subseteq \mathrm{Data}$$

where $\mathbf{r}(A)'$ is obtained from $\mathbf{r}(A)$ by replacing every $n+1$-ary $\mathbf{r}(\mathcal{L})$-predicate variable $\tilde{X}$ by $\mathrm{Data}' := \{(x, \mathbf{y}) \mid \mathrm{Data}(x)\}$. The proof is by induction on the structure of A. If A is an implication, or of the form $\nu\Phi(\mathbf{t})$, or $P(\mathbf{t})$ where P is a predicate constant, then A is non-computational, therefore $\mathbf{r}(A)' \subseteq \{()\} \subseteq \mathrm{Data}$. If A is $X(\mathbf{t})$, then $\mathbf{r}(A) = \{x \mid \tilde{X}(x, \mathbf{t})\}$, hence $\mathbf{r}(A)' = \{x \mid \mathrm{Data}'(x, \mathbf{t})\} = \mathrm{Data}$. The remaining cases are $\exists x\, A$, $\forall x\, A$, $A \vee B$, $A \wedge B$, and $\mu\Phi(\mathbf{t})$. The first four follow by a straightforward application of the induction hypotheses. In the last case it suffices to show $\mu\mathbf{r}(\Phi)' \subseteq \mathrm{Data}'$, which can be done by induction. We have to show $\mathbf{r}(\Phi)'(\mathrm{Data}') \subseteq \mathrm{Data}'$, i.e. $\forall x, \mathbf{y}\, (\mathbf{r}(\Phi(X))'(x, \mathbf{y}) \to \mathrm{Data}(x))$ where X is fresh. But this is the same as $\forall \mathbf{y}\, (\mathbf{r}(\Phi(X)(\mathbf{y}))' \subseteq \mathrm{Data}$. The latter follows from the (structural) induction hypothesis.

Theorem 5 (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 Sect. 4.

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

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

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 $a_i \in \mathrm{SD}$ such that $y = \Sigma_{i=0}^{\infty} 2^{-(i+1)} * a_i$.

In the third example we have

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

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 many writing nodes occur on each path through the tree ensuring that in the limit an infinite output stream is produced (see [Berger 2009] for details). The reading and writing nodes correspond to the absorption and emission of digits in [Edalat and Heckmann 2002].

4 Proof of the Soundness Theorem

The main task in proving the Soundness Theorem (Thm. 5) is to define the realisers of induction and coinduction and to prove their correctness.

We define program terms $\mathbf{map}_{X,A}$, $\mathbf{map}_{X,\mathcal{P}}$, $\mathbf{It}_{\mu X.\mathcal{P}}$, and $\mathbf{Coit}_{\nu X.\mathcal{P}}$, where X is a predicate variable, A is formula and $\mathcal{P}$ is a predicate, both strictly positive in X. In Lemma 9 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 $\mathbf{It}_{\mu X.\mathcal{P}}$ and $\mathbf{Coit}_{\nu X.\mathcal{P}}$ will be used to realise induction and coinduction. In [Miranda-Perea 2005] the iterators and coiterators are given as constants which expect map-terms as extra arguments, and the property stated in Lemma 9 is an assumption in the Soundness Theorem. Here, the terms $\mathbf{map}_{X,A}$, $\mathbf{map}_{X,\mathcal{P}}$, $\mathbf{It}_{\mu X.\mathcal{P}}$, and $\mathbf{Coit}_{\nu X.\mathcal{P}}$ 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.

$$\begin{aligned}\mathbf{map}_{X,A} &= \lambda f \lambda x\,.\,x \qquad \text{if } X \text{ is not free in } A, \text{ otherwise}\\ \mathbf{map}_{X,\mathcal{P}(\mathbf{t})} &= \mathbf{map}_{X,\mathcal{P}}\end{aligned}$$

$$
\begin{aligned}
\mathbf{map}_{X,A\vee B} &= \lambda f \lambda x\,.\,\mathrm{case}\,x\,\mathrm{of}\{\mathrm{inl}(y) \to \mathbf{map}_{X,A}\,f\,y\,;\,\mathrm{inr}(z) \to \mathbf{map}_{X,B}\,f\,z\}\\
\mathbf{map}_{X,A\wedge B} &= \mathbf{map}_{X,B} \quad \text{if } A \text{ is non-computational}\\
&= \mathbf{map}_{X,A} \quad \text{if } B \text{ is non-computational}\\
&= \lambda f \lambda x\,.\,\langle \mathbf{map}_{X,A} f\,(\pi_1(x)), \mathbf{map}_{X,B} f\,(\pi_2(x))\rangle \quad \text{otherwise}\\
\mathbf{map}_{X,A\to B} &= \mathbf{map}_{X,B} \quad \text{if } A \text{ is non-computational}\\
&= \lambda f \lambda g\,.\,\mathbf{map}_{X,B}\,f \circ g \quad \text{otherwise}
\end{aligned}
$$

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

$$
\begin{aligned}
\mathbf{It}_{\mu X.\mathcal{P}} &= \lambda s\,.\,\mathrm{rec}\,g\,.\,s \circ \mathbf{map}_{X,\mathcal{P}} g\\
\mathbf{Coit}_{\nu X.\mathcal{P}} &= \lambda s\,.\,\mathrm{rec}\,g\,.\,\mathbf{map}_{X,\mathcal{P}} g \circ s
\end{aligned}
$$

Lemma 6. *(a)* $\mathbf{It}_{\mu X.\mathcal{P}} s = s \circ \mathbf{map}_{X,\mathcal{P}}(\mathbf{It}_{\mu X.\mathcal{P}} s)$

(b) $\mathbf{Coit}_{\nu X.\mathcal{P}} s = \mathbf{map}_{X,\mathcal{P}}(\mathbf{Coit}_{\nu X.\mathcal{P}} s) \circ s$

(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

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

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

Lemma 7 (Substitution). $\mathbf{r}(\Phi)(\mathbf{r}(\mathcal{Q})) = \mathbf{r}(\Phi(\mathcal{Q}))$*, for every operator Φ and every computational predicate $\mathcal{Q}$.*

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 program terms. The following definitions will be used:

$$
\begin{aligned}
\mathcal{P} \circ f &:= \{(x, \mathbf{y}) \mid (f\,x, \mathbf{y}) \in \mathcal{P}\}\\
f * \mathcal{P} &:= \{(f\,x, \mathbf{y}) \mid (x, \mathbf{y}) \in \mathcal{P}\}
\end{aligned}
$$

Clearly, $(\mathcal{P} \circ f) \circ g = \mathcal{P} \circ (f \circ g)$ and $f * (g * \mathcal{P}) = (f \circ g) * \mathcal{P}$. The rationale for the first of the two definitions is that for computational 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 8 (Adjunction). $\mathcal{Q} \subseteq \mathcal{P} \circ f \;\Leftrightarrow\; f * \mathcal{Q} \subseteq \mathcal{P}$

Lemma 9 (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 computational $\mathcal{L}$-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 computational $\mathbf{r}(\mathcal{L})$-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 $X, \mathbf{Y}$ 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 $\Phi(X, \mathbf{Y})$. In the proof we allow ourselves to switch between (a) and (b) whenever convenient.

Case X does not occur freely in $\Phi(X, \mathbf{Y})$. Then $\mathbf{map}_{X,\Phi(X,\mathbf{Y})} f$ 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 $\Phi(X, \mathbf{Y})$. In particular, this implies that $\Phi(X, \mathbf{Y})$ is computational.

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

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

Case $\Phi(X, \mathbf{Y}) = \mu Z.\Phi_0(X, \mathbf{Y}, Z)$. Then $\mathbf{r}(\Phi)(\tilde{X}, \tilde{\mathbf{Y}}) = \mu\tilde{Z}.\mathbf{r}(\Phi_0)(\tilde{X}, \tilde{\mathbf{Y}}, \tilde{Z})$. 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

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

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

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

$$\begin{array}{cl}
 & \mathbf{r}(\Phi_0)(\mathcal{P}, \boldsymbol{\mathcal{Q}}, \mathcal{R} \circ \mathbf{map}_{X,\Phi(X,\mathbf{Y})} f) \\
\overset{\text{i.h.(c)}}{\subseteq} & \mathbf{r}(\Phi_0)(\mathcal{P}, \boldsymbol{\mathcal{Q}}, \mathcal{R}) \circ \mathbf{map}_{Z,\Phi_0(X,\mathbf{Y},Z)}(\mathbf{map}_{X,\Phi(X,\mathbf{Y})} f) \\
\overset{\text{i.h.(a)}}{\subseteq} & \mathbf{r}(\Phi_0)(\mathcal{Q}, \boldsymbol{\mathcal{Q}}, \mathcal{R}) \circ \mathbf{map}_{X,\Phi_0(X,\mathbf{Y},Z)} f \circ \mathbf{map}_{Z,\Phi_0(X,\mathbf{Y},Z)}(\mathbf{map}_{X,\Phi(X,\mathbf{Y})} f) \\
\overset{\text{Lem. 6 (c)}}{=} & \mathbf{r}(\Phi_0)(\mathcal{Q}, \boldsymbol{\mathcal{Q}}, \mathcal{R}) \circ \mathbf{map}_{X,\Phi(X,\mathbf{Y})} f \\
= & \mathbf{r}(\Phi_0)(\mathcal{Q}, \boldsymbol{\mathcal{Q}}, \mu\tilde{Z}.\mathbf{r}(\Phi_0)(\mathcal{P}, \boldsymbol{\mathcal{Q}}, \tilde{Z})) \circ \mathbf{map}_{X,\Phi(X,\mathbf{Y})} f \\
\overset{\text{Fixed P.}}{=} & \mu\tilde{Z}.\mathbf{r}(\Phi_0)(\mathcal{Q}, \boldsymbol{\mathcal{Q}}, \tilde{Z}) \circ \mathbf{map}_{X,\Phi(X,\mathbf{Y})} f \\
= & \mathcal{R} \circ \mathbf{map}_{X,\Phi(X,\mathbf{Y})} f
\end{array}$$

Case $\Phi(X, \mathbf{Y}) = \nu Z.\Phi_0(X, \mathbf{Y}, Z)$. Then $\mathbf{r}(\Phi)(\tilde{X}, \tilde{\mathbf{Y}}) = \nu\tilde{Z}.\mathbf{r}(\Phi_0)(\tilde{X}, \tilde{\mathbf{Y}}, \tilde{Z})$. Obviously, it is now more convenient to show (b). Assume $f * \mathcal{P} \subseteq \mathcal{Q}$. Setting $\mathcal{R} := \mathbf{r}(\Phi)(\mathcal{P}, \boldsymbol{\mathcal{Q}}) = \nu\tilde{Z}.\mathbf{r}(\Phi_0)(\mathcal{P}, \boldsymbol{\mathcal{Q}}, \tilde{Z})$, we have to show

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

We use coinduction on $\nu\tilde{Z}.\mathbf{r}(\Phi_0)(\mathcal{Q}, \boldsymbol{\mathcal{Q}}, \tilde{Z})$. The proof is exactly dual to the inductive proof above (using the structural induction hypothesis in the form (d) and (b)).

Proof of the Soundness Theorem (Thm. 5). 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 the axioms concerning inductively and coinductively defined computational predicates. Hence, let Φ be an operator such that $\mu\Phi$ (and hence $\nu\Phi$) is computational.

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

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

Induction. By the Substitution Lemma, we have

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

Therefore, in order to show that $\mathbf{It}_{\mu\varPhi}$ ($=: M$) realises induction, we assume

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

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

$$\begin{array}{rcl}
\mathbf{r}(\varPhi)(\mathbf{r}(\mathcal{Q}) \circ \mathbf{It}_{\mu\varPhi})s & \overset{\text{Map Lemma (c)}}{\subseteq} & \mathbf{r}(\varPhi)(\mathbf{r}(\mathcal{Q})) \circ \mathbf{map}_{\mu\varPhi}(\mathbf{It}_{\mu\varPhi} s) \\
& \overset{\text{assumption}}{\subseteq} & \mathbf{r}(\mathcal{Q}) \circ s \circ \mathbf{map}_{\mu\varPhi}(\mathbf{It}_{\mu\varPhi} s) \\
& \overset{\text{Lemma 6 (a)}}{=} & \mathbf{r}(\mathcal{Q}) \circ \mathbf{It}_{\mu\varPhi} s
\end{array}$$

Coinduction. Similar, using the Map Lemma (d) and Lemma 6 (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 $\mathrm{C}_0(\pi/4)$. Then the Soundness Theorem yields a program term S and a proof of $S\,\mathbf{r}\,\mathrm{C}_0(\pi/4)$. As 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 $\mathrm{C}_0(\pi/4)$, 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 with $|z| < 2^n$. The predicate $\mathbb{Z}$ can be defined inductively by

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

It easy to see that a realiser of $\mathbb{Z}(z, n)$ is a signed binary representation of z (that permits leading zeros). 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 Data(M), by the Data Lemma. Hence, all that remains to be shown is that whenever 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 standard domain-theoretic *denotational* semantics, which is of independent interest since it directly reflects the intuitive mathematical meaning of program terms. 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 would not have to be changed.

In the following we mean by a *domain* a *Scott-domain*, i.e. an algebraic, countably based, bounded complete, dcpo [Gierz et al. 2003]. 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 $[\![M]\!] : D^{\mathrm{Var}} \to D$. For example, $[\![\lambda x.M]\!]\xi = \mathrm{abst}(f)$ where $f(a) = [\![M]\!]\xi[x \mapsto a]$ and $[\![\mathrm{rec}\, x\,.\, M]\!]\xi$ is the least fixed point of f. Furthermore, if $[\![M]\!]\xi = \mathrm{abst}(f)$, then $[\![M\,N]\!]\xi = f([\![N]\!]\xi)$, otherwise the result is $\bot$.

If Ax is a set of non-computational $\mathcal{L}$-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 $[\![\mathrm{Data}]\!]$ of D such that

$$[\![\mathrm{Data}]\!] = \{()\} \cup \mathrm{inl}([\![\mathrm{Data}]\!]) \cup \mathrm{inr}([\![\mathrm{Data}]\!]) \cup \langle [\![\mathrm{Data}]\!], [\![\mathrm{Data}]\!] \rangle$$

Hence, if Data(M) is provable, then $[\![M]\!] \in [\![\mathrm{Data}]\!]$.

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 $c, c', \ldots$ range over closures and $v, v', \ldots$ range over values. We inductively define the relation $c \longrightarrow v$ (big-step reduction):

(i) $v \longrightarrow v$

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

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

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

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

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

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

Finally, in order to compute data we need a 'print' relation $c \Longrightarrow d$ between closures c and data terms d, which we define inductively as follows:

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

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

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

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

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

To every closure c we assign a term $\overline{c}$ 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 10 (Correctness). *(a) If $c \longrightarrow v$, then $\overline{c} = \overline{v}$ is provable.*

(b) If $c \Longrightarrow d$, then $\overline{c} = d$ 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 $c \Longrightarrow d$.

The Computational Adequacy Theorem below states that if a term M denotes a data d, then d can be computed from M.

Theorem 11 (Computational Adequacy). *If $[\![M]\!] = d$, then $(M, \emptyset) \Longrightarrow d$.*

The proof of this theorem is obtained by transferring Plotkin's Adequacy Theorem for PCF [Plotkin 1977] to the untyped setting by using a domain-theoretic variant of the reducibility or candidate method [Girard 1971, Tait 1975]. Similar methods were used before to prove computational adequacy for related calculi (e.g. [Amadio 1993] and [Winskel 1993]) and strong normalisation of λ-calculi with constants and rewrite rules ([Coquand and Spiwack 2006], [Berger 2008]). 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 $\mathbf{rk}(a_0) < \mathbf{rk}(\mathrm{abst}(f))$ 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 $\mathbf{rk}(a)$:

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

Alternatively, one could use the method in [Pitts 1994] to define similar "candidate" sets. Using (rk1) and (rk2) one can prove:

Lemma 12. *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 13. *$c \in \mathbf{Cl}(a)$ 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 $v = v'$.

Lemma 14. *If $c \in \mathbf{Cl}(d)$, where d is a data, then $c \Longrightarrow d$.*

Proof. Straightforward induction on d.

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 15 (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 12, $\eta(x) \in \mathbf{Cl}(a)$. By Lemma 13, 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 13.

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 $g(b) = [\![M]\!]^n\xi[x \mapsto b]$. 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 13, $(L, \eta[x \mapsto (M_0, \eta_0)]) \longrightarrow v)$ 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 13.

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 (rk2), 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 13, $(M_0, \eta_0[x \mapsto (N, \eta)]) \longrightarrow v$ 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 13 and Lemma 12.

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

Proof of the Adequacy Theorem (Thm. 11).

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 14.

6 Program extraction

The Soundness Theorem (Thm. 5) can be viewed as a general program extraction result since it states that from a proof of an arbitrary formula A one can extract a term M realizing A. The following theorem adds to this that if the realiser represented by M is an observable data, then this data can be *computed*.

Theorem 16 (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, $\mathrm{Data}(M)$ 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 10, $M = d$ is provable. It follows that $\mathbf{r}(A)(d)$ is provable.

Note that Theorem 16 is the best program extraction result one can hope for, because if the realiser is not a finite data, for example, an infinite stream, then it cannot be observed completely, but only finite initial segments of it can. We highlight this aspect by resuming our example from Sect. 5.

Lemma 17 (Printing digits).

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

Proof. Induction on $\mathbb{N}(n)$. Set $\mathcal{P} := \{n \mid \forall x\, (\mathrm{C}_0(x) \to \exists z\, (\mathbb{Z}(z, n) \wedge |x - \frac{z}{2^n}| \leq \frac{1}{2^n}))\}$. 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 takes as inputs a (unary) natural number n and a signed digit stream a representing some real number in $\mathbb{I}$, and computes a 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 first n 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.

We restricted ourselves to simple examples illustrating the method. More substantial applications are described in [Berger 2009]. 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 allow for a direct interpretation of realisers as programs in a call-by-name typed programming language.

A major piece of work that remains to be done is the implementation of the realisability interpretation in an interactive theorem prover in order to formally carry out the case studies that have so far been sketched on an informaly basis only. Furthermore, one needs to clarify the relation to other recent work on the theory and the implementation of inductive and coinductive definitions [Caffaglione and Gianantonio 2006], [Bertot 2007], [Niqui 2008], exact real number computation [Escardo and Marcial-Romero 2007], [Geuvers et al. 2007], [O'Connor and Spitters 2009], and realisability [Bauer and Stone 2007].

Acknowledgments

I would like to thank the referees for their constructive criticism and useful suggestions. I am particularly grateful to one referee who spotted an error in Lemma 4 of an earlier version of this paper.

References

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

[Amadio 1993] R. Amadio. On the adequacy of per models. In A.M. Borzyszkowski, S. Sokolowski, editors, Proc. Mathematical Foundations of Computer Science, volume 711 of *LNCS*, pages 222–231, 1993.

[Bauer and Blanck 2009] Andrej Bauer and Jens Blanck. Canonical effective subalgebras of classical algebras as constructive metric completions. In Andrej Bauer, Peter Hertling, and Ker-I Ko, editors, *6th Int'l Conf. on Computability and Complexity in Analysis*, Dagstuhl, Germany, 2009. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany.

[Bauer and Stone 2007] 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.

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

[Berger 2009] U. Berger. From coinductive proofs to exact real arithmetic. In E. Grädel and R. Kahle, editors, *Computer Science Logic*, volume 5771 of *LNCS*, pages 132–146. Springer, 2009.

[Bertot 2007] Y. Bertot. Affine functions and series with co-inductive real numbers. *Math. Struct. Comput. Sci.*, 17:37–63, 2007.

[Buchholz et al. 1981] 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.

[Bradfield and Stirling 2007] 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.

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

[Coquand and Spiwack 2006] 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.

[Edalat and Heckmann 2002] A. Edalat and R. Heckmann. Computing with real numbers: I. The LFT approach to real number computation; II. A domain framework for computational 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.

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

[Geuvers et al. 2007] H. Geuvers, M. Niqui, B. Spitters, and F. Wiedijk. Constructive analysis, types and exact real numbers. *Math. Struct. Comput. Sci.*, 17(1):3–36, 2007.

[Gierz et al. 2003] 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.

[Ghani et al. 2006] N. Ghani, P. Hancock, and D. Pattinson. Continuous functions on final coalgebras. *Electr. Notes in Theoret. Comput. Sci.*, 164, 2006.

[Girard 1971] 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.

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

[Lambek 1968] J. Lambek. A fixpoint theorem for complete categories. *Mathematische Zeitschrift*, 103:151–161, 1968.

[Matthes 2001] 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.

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

[Miranda-Perea 2005] F. Miranda-Perea. Realizability for monotone clausular (co)inductive definitions. *Electr. Notes in Theoret. Comput. Sci.*, 123:179–193, 2005.

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

[Niqui 2008] M. Niqui. Coinductive Formal Reasoning in Exact Real Arithmetic. *Logical Methods in Comp. Science*, 4(3):1–40, 2008.

[O'Connor and Spitters 2009] R. O'Connor and B. Spitters. A computer verified, monadic, functional implementation of the integral. To appear: Theoretical Computer Science. Available at `http://arxiv.org/abs/0809.1552`.

[Pitts 1994] A.M. Pitts. Computational adequacy via "mixed" inductive definitions. In *Proceedings of the 9th International Conference on Mathematical Foundations of Programming Semantics*, pages 72–82, London, UK, 1994. Springer-Verlag.

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

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

[Schwichtenberg 2009] H. Schwichtenberg. Realizability interpretation of proofs in constructive analysis. Theory Comput. Sys., to appear, 2009.

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

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

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

[Winskel 1993] G. Winskel. *The Formal Semantics of Programming Languages.* Foundations of Computing Series. The MIT Press, Cambridge, Massachusetts, 1993.