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 realisability interpretation of an extension of first-order predicate logic by least and greatest fixed points of strictly positive operators. The main 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* 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* 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 $\mathbb{I} := [-1,1] = \{x \mid -1 \le x \le 1\}$, $\mathrm{SD} := \{0,1,-1\}$, 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 $\{x \mid \exists i \in \mathrm{SD}, y \in \mathbb{I}\,(x = \mathrm{av}_i(y) \wedge X(y))\}$. Classically, 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

Andrej Bauer, Peter Hertling, Ker-I Ko (Eds.)
6th Int'l Conf. on Computability and Complexity in Analysis, 2009, pp. 49-60
http://drops.dagstuhl.de/opus/volltexte/2009/2258

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 will be worked in a forthcoming publication.

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. 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]. 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 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 [Sch09] and Hernest and Oliva [HO08] which allow for a fine control of the amount of computational information extracted from proofs.

We state most of the results without proof. Full proofs will be given in an extended version of this paper.

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 $\{\boldsymbol{x} \mid \mathcal{P}(\boldsymbol{x}) \wedge \mathcal{Q}(\boldsymbol{x})\}$, etc. 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 ones for intuitionistic predicate calculus with equality. In addition, we have the axioms

$$\begin{array}{lll} \text{Closure} & \Phi(\mu\Phi) \subseteq \mu\Phi & \qquad \text{Induction} \quad \Phi(\mathcal{Q}) \subseteq \mathcal{Q} \to \mu\Phi \subseteq \mathcal{Q} \\ \text{Coclosure} & \nu\Phi \subseteq \Phi(\nu\Phi) & \qquad \text{Coinduction} \quad \mathcal{Q} \subseteq \Phi(\mathcal{Q}) \to \mathcal{Q} \subseteq \nu\Phi \end{array}$$

In addition we allow any axioms expressible by non-computational formulas that hold in the intended model. In particular, it is possible to add all classical non-computational tautologies as axioms such as, for example, $\exists x\, A \leftrightarrow \neg\forall x\, \neg A$ for non-computational A. 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. The following basic facts are easy to prove.

Lemma 1. *(a) If $\Gamma(X) \vdash A(X)$, then $\Gamma(\mathcal{P}) \vdash A(\mathcal{P})$.*
(b) If $\Gamma \vdash \Phi(X) \subseteq \Psi(X)$, then $\Gamma \vdash \mu\Phi \subseteq \mu\Psi$ and $\Gamma \vdash \nu\Phi \subseteq \nu\Psi$.
(c) $\mathcal{P} \subseteq \mathcal{Q} \to \Phi(\mathcal{P}) \subseteq \Phi(\mathcal{Q})$.
(d) $\Phi(\mu\Phi) = \mu\Phi$ and $\Phi(\nu\Phi) = \nu\Phi$.

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 $\mathrm{inl}(\cdot)$ $\mathrm{inr}(\cdot)$. 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

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

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. In the definitions we take special care of non-computational formulas and predicates in order to get optimised realisers. If A is non-computational, then $\mathbf{r}(A) = \{() \mid A\}$. If $\mathcal{P}$ is non-computational, then $\mathbf{r}(\mathcal{P}) = \{((), \boldsymbol{x}) \mid \mathcal{P}(\boldsymbol{x})\}$. In all other cases:

$$\begin{aligned}
\mathbf{r}(\mathcal{P}(\boldsymbol{t})) &= \{x \mid \mathbf{r}(\mathcal{P})(x, \boldsymbol{t})\} & \mathbf{r}(A \to B) &= \{f \mid f(\mathbf{r}(A)) \subseteq \mathbf{r}(B)\}\\
\mathbf{r}(A \vee B) &= \mathrm{inl}(\mathbf{r}(A)) \cup \mathrm{inl}(\mathbf{r}(B)) & \mathbf{r}(A \wedge B) &= \langle \mathbf{r}(A), \mathbf{r}(B)\rangle\\
\mathbf{r}(\exists y\, A) &= \{x \mid \exists y\, (\mathbf{r}(A)(x))\} & \mathbf{r}(\forall y\, A) &= \{x \mid \forall y\, (\mathbf{r}(A)(x))\}\\
\mathbf{r}(X) &= \tilde{X} & \mathbf{r}(\{\boldsymbol{x} \mid A\}) &= \{(y, \boldsymbol{x}) \mid \mathbf{r}(A)(\boldsymbol{x})\}\\
\mathbf{r}(\mu X.\mathcal{P}) &= \mu\tilde{X}.\mathbf{r}(\mathcal{P}) & \mathbf{r}(\nu X.\mathcal{P}) &= \nu\tilde{X}.\mathbf{r}(\mathcal{P})
\end{aligned}$$

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

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

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

Lemma 2 (Data formulas). $\mathbf{r}(A) \subseteq \mathrm{Data}$ *for every data formula* A.

Proof. One shows more generally: if A is a formula such that every subformula which is an implication or of the form $\nu\Phi(\boldsymbol{t})$ is non-computational, then $\mathbf{r}(A)(\mathbf{Data}') \subseteq \mathrm{Data}$, where $\mathbf{r}(A)(\mathbf{Data}')$ is obtained from $\mathbf{r}(A)$ by replacing every $n+1$-ary $\mathbf{r}(\mathcal{L})$-predicate variable $\tilde{X}$ by the predicate $\mathrm{Data}' := \{(x, \boldsymbol{y}) \mid \mathrm{Data}(x)\}$. The easy proof is by induction on the structure of A.

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

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

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

In the third example we have

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

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.

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 $\mathbf{map}_{X,A}$, $\mathbf{map}_{X,\mathcal{P}}$, $\mathbf{It}_{\mathrm{fix}\,X\,.\,\mathcal{P}}$, and $\mathbf{Coit}_{\mathrm{fix}\,X\,.\,\mathcal{P}}$, where X is a predicate variable, A is formula and $\mathcal{P}$ is a predicate, both strictly positive in X. In Lemma 6 we will show that $\mathbf{map}_{X,\mathcal{P}}$ realises the monotonicity of $\mathcal{P}$ w.r.t. X. The terms $\mathbf{It}_{\mathrm{fix}\,X\,.\,\mathcal{P}}$ and $\mathbf{Coit}_{\mathrm{fix}\,X\,.\,\mathcal{P}}$ 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 6 is an assumption in the Soundness Theorem.

Here, the terms $\mathbf{map}_{X,A}$, $\mathbf{map}_{X,\mathcal{P}}$, $\mathbf{It}_{\mathrm{fix}\,X\,.\,\mathcal{P}}$, and $\mathbf{Coit}_{\mathrm{fix}\,X\,.\,\mathcal{P}}$ 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. $\mathbf{map}_{X,A} = \mathbf{map}_{X,\mathcal{P}} = \lambda f \lambda x\,.\,x$ if X is not free in A or $\mathcal{P}$. Otherwise:

$$
\begin{aligned}
\mathbf{map}_{X,\mathcal{P}(\boldsymbol{t})} &= \mathbf{map}_{X,\mathcal{P}}\\
\mathbf{map}_{X,A\wedge B} &= \lambda f \lambda x\,.\,\langle \mathbf{map}_{X,A} f\,(\pi_1(x)), \mathbf{map}_{X,B} f\,(\pi_2(x))\rangle\\
\mathbf{map}_{X,A\vee B} &= \lambda f \lambda x\,.\,\mathrm{case}\,x\,\mathrm{of}\{\mathrm{inl}(y) \to \mathbf{map}_{X,A}\,f\,y\,;\,\mathrm{inr}(z) \to \mathbf{map}_{X,B}\,f\,z\}\\
\mathbf{map}_{X,A\to B} &= \lambda f \lambda g\,.\,\mathbf{map}_{X,B}\,f \circ g\\
\mathbf{map}_{X,\{\boldsymbol{x}|A\}} &= \mathbf{map}_{X,A}\\
\mathbf{map}_{X,X} &= \lambda f\,.\,f\\
\mathbf{map}_{X,\mu Y.\mathcal{P}} &= \lambda f\,.\,\mathbf{It}_{\mathrm{fix}\,Y\,.\,\mathcal{P}}(\mathbf{map}_{X,\mathcal{P}} f)\\
\mathbf{map}_{X,\nu Y.\mathcal{P}} &= \lambda f\,.\,\mathbf{Coit}_{\mathrm{fix}\,Y\,.\,\mathcal{P}}(\mathbf{map}_{X,\mathcal{P}} f)\\
\mathbf{It}_{\mathrm{fix}\,X\,.\,\mathcal{P}} &= \lambda s\,.\,\mathrm{rec}\,g\,.\,s \circ \mathbf{map}_{X,\mathcal{P}} g\\
\mathbf{Coit}_{\mathrm{fix}\,X\,.\,\mathcal{P}} &= \lambda s\,.\,\mathrm{rec}\,g\,.\,\mathbf{map}_{X,\mathcal{P}} g \circ s
\end{aligned}
$$

Lemma 3. *(a)* $\mathbf{It}_{\mathrm{fix}\,X\,.\,\mathcal{P}} s = s \circ \mathbf{map}_{X,\mathcal{P}}(\mathbf{It}_{\mathrm{fix}\,X\,.\,\mathcal{P}} s)$
(b) $\mathbf{Coit}_{\mathrm{fix}\,X\,.\,\mathcal{P}} s = \mathbf{map}_{X,\mathcal{P}}(\mathbf{Coit}_{\mathrm{fix}\,X\,.\,\mathcal{P}} s) \circ s$
(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. Easy calculation using the equational axioms for the calculus.

Lemma 4 (Substitution). $\mathbf{r}(\Phi)(\mathbf{r}(\mathcal{Q})) = \mathbf{r}(\Phi(\mathcal{Q}))$ *for every operator* Φ *and 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 predicate terms. The following definitions will be used:

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

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

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 5 (Adjunction). $\mathcal{Q} \subseteq \mathcal{P} \circ f \;\Leftrightarrow\; f * \mathcal{Q} \subseteq \mathcal{P}$

Lemma 6 (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. We only carry out in detail the difficult cases, namely when Φ is defined by induction or coinduction.

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 induct on $\mu\tilde{Z}.\mathbf{r}(\Phi_0)(\mathcal{P}, \boldsymbol{\mathcal{Q}}, \tilde{Z})$. Hence, we have to show $\mathbf{r}(\Phi_0)(\mathcal{P}, \boldsymbol{\mathcal{Q}}, \mathcal{R} \circ \mathbf{map}_{X,\Phi(X,\boldsymbol{Y})} f) \subseteq \mathcal{R} \circ \mathbf{map}_{X,\Phi(X,\boldsymbol{Y})} f$.

$$\begin{array}{rl}
 & \mathbf{r}(\Phi_0)(\mathcal{P}, \boldsymbol{\mathcal{Q}}, \mathcal{R} \circ \mathbf{map}_{X,\Phi(X,\boldsymbol{Y})} f) \\
\overset{\text{i.h.(c)}}{\subseteq} & \mathbf{r}(\Phi_0)(\mathcal{P}, \boldsymbol{\mathcal{Q}}, \mathcal{R}) \circ \mathbf{map}_{Z,\Phi_0(X,\boldsymbol{Y},Z)}(\mathbf{map}_{X,\Phi(X,\boldsymbol{Y})} f) \\
\overset{\text{i.h.(a)}}{\subseteq} & \mathbf{r}(\Phi_0)(\mathcal{Q}, \boldsymbol{\mathcal{Q}}, \mathcal{R}) \circ \mathbf{map}_{X,\Phi_0(X,\boldsymbol{Y},Z)} f \circ \mathbf{map}_{Z,\Phi_0(X,\boldsymbol{Y},Z)}(\mathbf{map}_{X,\Phi(X,\boldsymbol{Y})} f) \\
\overset{\text{L. 3 (c)}}{=} & \mathbf{r}(\Phi_0)(\mathcal{Q}, \boldsymbol{\mathcal{Q}}, \mathcal{R}) \circ \mathbf{map}_{X,\Phi(X,\boldsymbol{Y})} f \\
= & \mathbf{r}(\Phi_0)(\mathcal{Q}, \boldsymbol{\mathcal{Q}}, \mu\tilde{Z}.\mathbf{r}(\Phi_0)(\mathcal{P}, \boldsymbol{\mathcal{Q}}, \tilde{Z})) \circ \mathbf{map}_{X,\Phi(X,\boldsymbol{Y})} f \\
\overset{\text{Fixed P.}}{=} & \mu\tilde{Z}.\mathbf{r}(\Phi_0)(\mathcal{Q}, \boldsymbol{\mathcal{Q}}, \tilde{Z}) \circ \mathbf{map}_{X,\Phi(X,\boldsymbol{Y})} f \\
= & \mathcal{R} \circ \mathbf{map}_{X,\Phi(X,\boldsymbol{Y})} f
\end{array}$$

Case $\Phi(X,\boldsymbol{Y}) = \nu Z.\Phi_0(X,\boldsymbol{Y},Z)$. Then $\mathbf{r}(\Phi)(\tilde{X},\tilde{\boldsymbol{Y}}) = \nu\tilde{Z}.\mathbf{r}(\Phi_0)(\tilde{X},\tilde{\boldsymbol{Y}},\tilde{Z})$. Obviously, it is now more convenient to show (b). Assume $f{*}\mathcal{P} \subseteq \mathcal{Q}$. Setting $\mathcal{R} := \mathbf{r}(\Phi)(\mathcal{P},\boldsymbol{\mathcal{Q}}) = \nu\tilde{Z}.\mathbf{r}(\Phi_0)(\mathcal{P},\boldsymbol{\mathcal{Q}},\tilde{Z})$ we use coinduction to show $\mathbf{map}_{X,\Phi(X,\boldsymbol{Y})}f * \mathcal{R} \subseteq \nu\tilde{Z}.\mathbf{r}(\Phi_0)(\mathcal{Q},\boldsymbol{\mathcal{Q}},\tilde{Z})$. The proof is exactly dual to the inductive proof above (using the 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 $B_1,\dots,B_n \vdash A$ one can extract a program term M with free variables among $x_1,\dots,x_n$ such that $\mathbf{r}(B_1)(x_1),\dots,\mathbf{r}(B_n)(x_n) \vdash \mathbf{r}(A)(M)$. The only interesting cases are induction and coinduction.

Induction. By the Substitution Lemma, we have

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

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

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

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

$$\begin{array}{lcl}
\mathbf{r}(\Phi)(\mathbf{r}(\mathcal{Q}) \circ \mathbf{It}_{\varphi(\alpha)})s & \overset{\text{Map Lemma (c)}}{\subseteq} & \mathbf{r}(\Phi)(\mathbf{r}(\mathcal{Q})) \circ \mathbf{map}_{\varphi(\alpha)}(\mathbf{It}_{\varphi(\alpha)}s) \\
& \overset{\text{assumption}}{\subseteq} & \mathbf{r}(\mathcal{Q}) \circ s \circ \mathbf{map}_{\varphi(\alpha)}(\mathbf{It}_{\varphi(\alpha)}s) \\
& \overset{\text{Lemma 3 (a)}}{=} & \mathbf{r}(\mathcal{Q}) \circ \mathbf{It}_{\varphi(\alpha)}s
\end{array}$$

Coinduction. Similar, using the Map Lemma (d) and Lemma 3 (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 to compute data. 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.

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"). 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 $\mathrm{Data}(M)$ 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):

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

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

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

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

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

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

$$\frac{c \longrightarrow ((), \eta)}{c \Longrightarrow ()} \qquad \frac{c \longrightarrow (\mathrm{inl}(M), \eta) \qquad (M, \eta) \Longrightarrow d}{c \Longrightarrow \mathrm{inl}(d)}\ \text{sim. inr}(M)$$

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

The inductive definition of $c \Longrightarrow d$ gives rise to an algorithm computing d from c in a call-by-name fashion. It follows 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 term $\overline{c}$ by 'flattening', i.e. removing the structure provided by the nested environments: $\overline{(M,\eta)} = M[\overline{\eta(x)}/x \mid x \in \mathrm{dom}(\eta)]$.

Lemma 7 (Correctness).

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

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

The proof of the Adequacy Theorem is uses a technique that has been used for a similar purpose in [Win93] and [CS06]. It can be viewed as transformation of Plotkin's Adequacy Theorem for PCF [Plo77] to the untyped setting. 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, which 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 $\mathbf{rk}(\cdot) : D_0 \to \mathbb{N}$ satisfying:

(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. 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 $\mathbf{rk}(a)$: $\mathbf{Cl}(\bot)$ is the set of all closures, otherwise

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

Alternatively, one could use Pitt's method [Pit94] to define similar "candidate" sets. Using (rk1) and (rk2) one can prove:

Lemma 8. *(a) If a, b are compact with $a \sqsubseteq b$, then $\mathbf{Cl}(a) \supseteq \mathbf{Cl}(b)$.*
(b) $c \in \mathbf{Cl}(a)$ iff there exists a value v with $c \longrightarrow v$ and $v \in \mathbf{Cl}(a)$.
(c) If $c \in \mathbf{Cl}(d)$, where d is a data, then $c \Longrightarrow d$.

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 9 (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 by induction on n that if $\eta \in \mathbf{Cl}(\xi)$ and a is compact with $a \sqsubseteq [\![M]\!]^n\xi$, then $(M, \eta) \in \mathbf{Cl}(a)$.

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 8 (c).

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 2, $\mathrm{Data}(M)$ 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 7, $M = d$ is provable. It follows that $\mathbf{r}(A)(d)$ is provable.

6 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 to be published in forthcoming papers. 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 coinductive definitions and proofs [CDG06,Ber07], exact real arithmetic [MRE07,GNSW07,EH02,Sch09], realisability [BS07], 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.

[Ber07] Y. Bertot. Affine functions and series with co-inductive real numbers. *Math. Struct. Comput. Sci.*, 17:37–63, 2007.

[BS07] 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*, volume 4497 of *LNCS*, pages 28–42, 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 *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 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.

[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. *Electr. Notes in Theoret. Comp. Sci.*, 164, 2006.

[GNSW07] 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.

[HO08] 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.

[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 definitions. *Electron. Notes Theoret. Comput. Sci.*, 123:179–193, 2005.

[MRE07] 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.

[Pit94] A.M. Pitts. Computational adequacy via "mixed" inductive definitions. In *Proc. 9th International Conference on Mathematical Foundations of Programming Semantics*, pages 72–82, London, UK, 1994. Springer-Verlag.

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

[Sch09] H. Schwichtenberg. Realizability interpretation of proofs in constructive analysis. Theory Comput. Sys., to appear, 2009.

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

[Win93] G. Winskel. *The Formal Semantics of Programming Languages*. Found. of Comput. Series. The MIT Press, Cambridge, Massachusetts, 1993.