Computational Adequacy via 'Mixed' Inductive Definitions

Andrew M. Pitts*

University of Cambridge Computer Laboratory,
Pembroke Street, Cambridge CB2 3QG, England

Abstract. For programming languages whose denotational semantics uses fixed points of domain constructors of mixed variance, proofs of correspondence between operational and denotational semantics (or between two different denotational semantics) often depend upon the existence of relations specified as the fixed point of non-monotonic operators. This paper describes a new approach to constructing such relations which avoids having to delve into the detailed construction of the recursively defined domains themselves. The method is introduced by example, by considering the proof of computational adequacy of a denotational semantics for expression evaluation in a simple, untyped functional programming language.

1 Introduction

It is well known that various domain constructors can be extended to act on *relations* on domains. For example, given binary relations R and S on domains D and E, there is a binary relation $R \to S$ on the domain of continuous functions $D \to E$ given by: $(f, g) \in (R \to S)$ if and only if for all $(x, y) \in R$, $(f(x), g(y)) \in S$. The utility of such constructions on relations can be seen in the various applications of 'logical relations' techniques in denotational semantics, pioneered by Milne [6], Plotkin [10, 11] and Reynolds [12]. For applications to programming language semantics, undoubtedly the most important domain-construction technique is that of solving recursive domain equations. In general, the body of a domain equation may involve not only positive, but also negative occurrences of the defined domain. Traditionally, the construction of the action on relations of such a recursively defined domain constructor has involved delving into the quite heavy technical machinery used to establish the existence of the domain itself. In [9] the author described a more elementary method of construction, inspired by Freyd's recent categorical analysis of recursive types [1, 2, 3]. It makes use of mixed inductive/co-inductive definitions. Apart from this, only quite straightforward domain-theoretic techniques are needed—namely fixed point induction and the fact that the identity function on a recursively defined domain is the

* Research supported by UK SERC grant GR/G53279, CEC ESPRIT project CLICS-II and CEC SCIENCE project PL910296

least fixed point of a certain continuous functional canonically associated with the domain equation.

In this paper, we illustrate the use of this new method of construction of relations on recursively defined domains by example. We consider a specific application where such relations are needed—namely the proof of correspondence between the denotational and operational semantics of a functional programming language. Recall that a denotational semantics is called 'computationally adequate' for an operationally defined expression evaluator provided any expression evaluates to canonical form just in case its denotation is not the bottom element of the corresponding semantic domain. This property is important since, combined with compositionality of the denotational semantics, it implies that observational equivalence of programming language expressions may be established via equality of denotations. See Meyer [5] for a discussion of this property. Proofs of computational adequacy are non-trivial when the denotational semantics of the programming language involves solving recursive domain equations $X = \Phi(X)$ in which X occurs negatively (and maybe also positively) in the domain constructor $\Phi(X)$. We consider a very simple example of this — an untyped lambda calculus — in order not to obscure the novelty of our approach with language-related details.

The computational adequacy property is reviewed in Sect. 2, where we recall how it can be established via the existence of a certain recursively specified relation of 'formal approximation' between domain elements and programs. Our new method of construction of the formal approximation relation $\lhd$ is given in Sect. 3. The method involves three steps:

- First, the negative and positive occurrences of $\lhd$ in the body of its recursive specification $\lhd = \phi(\lhd)$ are replaced by fresh variables $\lhd^-$ and $\lhd^+$ respectively. This results in a new operator $\psi(\lhd^-, \lhd^+)$ which is monotonic in $\lhd^+$, anti-monotonic in $\lhd^-$, and from which the original operator ϕ can be obtained by diagonalizing. (This separation of variables is a key feature of Freyd's recent analysis of recursive types.)
- Secondly, the new operator ψ is used to give simultaneous inductive definitions of positive and negative versions of the formal approximation relation.
- Lastly, these positive and negative versions are proved equal, and so by construction constitute the required relation. The proof of equality is a simple fixed point induction argument. It makes use of a key property of recursively defined domains, namely that they are 'minimal invariants' for their associated domain constructor: see Definition 2.

Finally in Sect. 4 we indicate an important aspect of the above method of construction, namely that it not only produces a suitable relation, but also characterizes it via a 'universal property' (in the category-theoretic sense). It is this universal property which gives rise to the reasoning principles established in [8, 9].

2 Computational adequacy

In this section we review the standard approach to proving computational adequacy, using a very simple untyped functional programming language $\mathcal{L}$ to illustrate what is involved. $\mathcal{L}$ is an untyped version of Plotkin's call-by-name PCF [10]. Its expressions are given by:

$$\begin{array}{lll} M ::= x & & \text{variables} \\ \quad | \ \underline{n} & & \text{numerals} \\ \quad | \ \mathit{suc}(M) & & \text{successor} \\ \quad | \ \mathit{pred}(M) & & \text{predecessor} \\ \quad | \ \mathit{if}\ M = 0\ \mathit{then}\ M\ \mathit{else}\ M & & \text{conditional} \\ \quad | \ \lambda x.M & & \text{function abstraction} \\ \quad | \ MM & & \text{function application} \end{array}$$

where x runs over a fixed, infinite set of *variables*, and n runs over the set of integers, $\mathbb{Z}$. Function abstraction is the only variable-binding construct (occurrences of x in M are bound in $\lambda x.M$). We denote by $M[M'/x]$ the result of substituting an expression M' for all free occurrences of x in M (subject to the usual conventions about renaming bound variables if necessary to avoid variable capture).

Let *Prog* ('programs') denote the collection of closed expressions in $\mathcal{L}$, i.e. those with no free variables. We denote by *Val* ('values') the subset of *Prog* consisting of all *canonical forms*, which here means all closed expressions that are either numerals $\underline{n}$ or function abstractions $\lambda x.M$. An operational semantics for $\mathcal{L}$ can be given via an *evaluation relation*

$$P \Downarrow V \qquad (P \in \mathit{Prog}, V \in \mathit{Val})$$

which is the subset of $\mathit{Prog} \times \mathit{Val}$ inductively defined by the rules in Table 1. The last rule embodies the non-strict, or 'call-by-name' scheme for evaluating function applications.

Table 1. Rules for evaluating programs in $\mathcal{L}$.

$$\frac{}{V \Downarrow V} \qquad \frac{P \Downarrow \underline{n}}{\mathit{suc}(P) \Downarrow \underline{n+1}} \qquad \frac{P \Downarrow \underline{n+1}}{\mathit{pred}(P) \Downarrow \underline{n}}$$

$$\frac{P \Downarrow \underline{0} \qquad Q \Downarrow V}{(\mathit{if}\ P = 0\ \mathit{then}\ Q\ \mathit{else}\ R) \Downarrow V} \qquad \frac{P \Downarrow \underline{n} \qquad R \Downarrow V}{(\mathit{if}\ P = 0\ \mathit{then}\ Q\ \mathit{else}\ R) \Downarrow V}\ (n \neq 0)$$

$$\frac{P \Downarrow \lambda x.M \qquad M[Q/x] \Downarrow V}{PQ \Downarrow V}$$

Denotational semantics for expressions in $\mathcal{L}$ can be given using a solution to the domain equation

$$D \cong (\mathbb{Z} + (D \to D))_\perp \quad . \tag{1}$$

Here we can take 'domain' to mean a partially ordered set with a least element $\perp$ and possessing least upper bounds $\bigsqcup_{i<\omega} d_i$ of all countable chains $d_0 \sqsubseteq d_1 \sqsubseteq \cdots$. The domain on the right-hand side of (1) is the lift of the disjoint union of the set of integers $\mathbb{Z}$ (discretely ordered) with the domain of continuous functions $D \to D$ (ordered pointwise). Thus a domain D is a solution to (1) if it comes equipped with continuous functions

$$\begin{aligned} \mathrm{num} &: \mathbb{Z} \longrightarrow D \\ \mathrm{fun} &: (D \to D) \longrightarrow D \end{aligned}$$

which combine to give an order isomorphism between the disjoint union $\mathbb{Z} + (D \to D)$ and $\{d \in D \mid d \neq \perp\}$. Given such a D, one can assign to each $\mathcal{L}$-expression M and each environment ρ (a finite partial function from the set of variables to D) whose domain of definition contains the free variables of M, an element

$$[\![M]\!]\rho \in D \quad .$$

The definition of $[\![M]\!]\rho$ is by induction on the structure of M and is quite standard; for the record, we give the clauses of the definition in Table 2. The clause for $\lambda x.M$ uses the notation $\rho[x \mapsto d]$ to indicate the environment mapping x to d and otherwise acting like ρ.

If an environment ρ' extends ρ, then $[\![M]\!]\rho' = [\![M]\!]\rho$. In particular for programs $P \in \mathit{Prog}$, i.e. for closed expressions, $[\![P]\!]\rho$ is an element of D which is independent of ρ, and which we write simply as $[\![P]\!]$. The following property can be established by induction on the derivation of the evaluation $P \Downarrow V$.

Proposition 1 (Soundness). *If $P \Downarrow V$ then $[\![P]\!] = [\![V]\!]$.*

Of course one cannot expect the converse of this soundness property to hold, since function abstractions are canonical forms whether or not the body of the abstraction is fully evaluated. For example $[\![\lambda x.\mathit{suc}(\underline{0})]\!] = [\![\lambda x.\underline{1}]\!]$, but $\lambda x.\mathit{suc}(\underline{0}) \Downarrow \lambda x.\underline{1}$ does not hold. However, if $[\![P]\!] = [\![V]\!]$, then since (from Table 2) the denotations of canonical forms are non-bottom elements of D, one at least has that $[\![P]\!] \neq \perp$. D is called *computationally adequate* if for all programs P, $[\![P]\!] \neq \perp$ holds (if and) only if $P \Downarrow V$ holds for some canonical form V. The point of this property is that it permits observational equivalence of $\mathcal{L}$-expressions to be established via equality of denotations: see Meyer [5].

Whilst the soundness property of Proposition 1 holds for any domain D which is a solution for the domain equation (1), computational adequacy only holds if D is a suitably minimal solution. One way of expressing this minimality, essentially due to D. Scott, is as follows.

Table 2. Denotations of $\mathcal{L}$-expressions.

$$[\![x]\!]\rho = \rho(x)$$

$$[\![\underline{n}]\!]\rho = \mathrm{num}(n)$$

$$[\![suc(M)]\!]\rho = \begin{cases} \mathrm{num}(n+1) & \text{if } [\![M]\!]\rho = \mathrm{num}(n) \\ \bot & \text{otherwise} \end{cases}$$

$$[\![pred(M)]\!]\rho = \begin{cases} \mathrm{num}(n-1) & \text{if } [\![M]\!]\rho = \mathrm{num}(n) \\ \bot & \text{otherwise} \end{cases}$$

$$[\![\mathit{if}\ M = 0\ \mathit{then}\ M'\ \mathit{else}\ M'']\!]\rho = \begin{cases} [\![M']\!]\rho & \text{if } [\![M]\!]\rho = \mathrm{num}(0) \\ [\![M'']\!]\rho & \text{if } [\![M]\!]\rho = \mathrm{num}(n) \text{ and } n \neq 0 \\ \bot & \text{otherwise} \end{cases}$$

$$[\![\lambda x.M]\!]\rho = \mathrm{fun}(\lambda d \in D.[\![M]\!]\rho[x \mapsto d])$$

$$[\![MM']\!]\rho = \begin{cases} f([\![M']\!]\rho) & \text{if } [\![M]\!]\rho = \mathrm{fun}(f) \\ \bot & \text{otherwise} \end{cases}$$

Definition 2 (Minimal invariant property). Let $\Phi(-) \stackrel{def}{=} (\mathbb{Z} + (-) \to (-))_\bot$. An *invariant* for Φ is a domain D equipped with an order isomorphism $i : D \cong \Phi(D)$. Such an invariant is *minimal* if the identity function $id_D \in (D \to D)$ is the least fixed point of the continuous function $\delta_\Phi : (D \to D) \longrightarrow (D \to D)$ which maps $e \in (D \to D)$ to $i^{-1}\Phi(e)i$. Here $\Phi(e) : \Phi(D) \longrightarrow \Phi(D)$ is the function which is the identity on $\bot$ and integers, and acts on functions by pre- and post-composing with e. Thus if the isomorphism i is described in terms of functions $\mathrm{num} : \mathbb{Z} \longrightarrow D$ and $\mathrm{fun} : (D \to D) \longrightarrow D$ as above, then

$$\delta_\Phi(e)(d) = \begin{cases} \mathrm{num}(n) & \text{if } d = \mathrm{num}(n) \\ \mathrm{fun}(e \circ f \circ e) & \text{if } d = \mathrm{fun}(f) \\ \bot & \text{if } d = \bot \end{cases} \tag{2}$$

for all $e \in (D \to D)$ and all $d \in D$.

Theorem 3 (Computational Adequacy). *If (D, i) is a minimal invariant for $(\mathbb{Z} + (-) \to (-))_\bot$, then the denotational semantics of $\mathcal{L}$ in D is computationally adequate, i.e. for all $P \in Prog$*

$$\exists V (P \Downarrow V) \Leftrightarrow [\![P]\!] \neq \bot \ .$$

The statement of this theorem appears more general than corresponding results in the literature, which refer to the computational adequacy of a particular domain. However it is not really so general, since one can show that

– the minimal invariant property characterizes solutions to domain equations uniquely up to isomorphism; and
– the solutions to domain equations $D \cong \Phi(D)$ (for a wide class of domain constructors $\Phi(-)$) constructed via any of the several methods available in the literature (such as via colimits of embedding-projection pairs: see [4, Sect. 10.1]; or via Scott's 'information systems': see [15, Chap. 12]) yield minimal invariants. Indeed, the minimal invariant property amounts to the fact, familiar from the 'local' characterization of colimits of chains of embeddings [14, Theorem 2], that any element d of a recursively defined domain $\mathrm{rec}X.\Phi(X)$ can be expressed as the least upper bound of a chain of projections of the element:

$$d = \bigsqcup_{i<\omega} \pi_i(d), \qquad \text{where } \begin{cases} \pi_0(d) = \bot \\ \pi_{i+1}(d) = \delta_\Phi(\pi_i)(d) \end{cases} .$$

However, it seems a step forward to have an abstract criterion on solutions of domain equations that suffices for computational adequacy. Moreover, the key construction needed in the new proof of Theorem 3 which we give in the next section, relies directly upon the minimal invariant property of D rather than upon any particular concrete construction of the domain.

The classical method for proving Theorem 3 is an adaptation by Milne [6] and Plotkin [10, 11] of Tait's use of 'computability' predicates in normalization proofs. It relies upon the construction of a binary relation between domain elements and programs with the following properties.

Definition 4. Let D be a solution to (1). A *formal approximation* relation is a binary relation $\lhd \subseteq D \times \mathit{Prog}$ satisfying:

1. For all $d \in D$ and $P \in \mathit{Prog}$, $d \lhd P$ if and only if
   either $d = \bot$,
   or $d = \mathrm{num}(n)$ for some n such that $P \Downarrow \underline{n}$,
   or $d = \mathrm{fun}(f)$ and $P \Downarrow \lambda x.M$ for some f and $\lambda x.M$ such that for all d', P', if $d' \lhd P'$ then $f(d') \lhd M[P'/x]$.
2. If $d_0 \sqsubseteq d_1 \sqsubseteq d_2 \sqsubseteq \cdots$ is a chain in D with $d_i \lhd P$ for all i, then $\bigsqcup_{i<\omega} d_i \lhd P$.

Given such a formal approximation relation, for any expression M, any environment ρ whose domain of definition $\mathit{dom}(\rho) = \{x_1, \ldots, x_n\}$ contains the free variables in M, and any programs $P_1, \ldots, P_n$, it is easy to prove by induction on the structure of M that

$$\rho(x_1) \lhd P_1 \wedge \cdots \wedge \rho(x_n) \lhd P_n \Rightarrow [\![M]\!]\rho \lhd M[P_1/x_1, \ldots, P_n/x_n] .$$

In particular, in case $n = 0$ we obtain for all programs P that

$$[\![P]\!] \lhd P .$$

Hence if $[\![P]\!] \neq \bot$, then by the properties of $\lhd$ in part 1 of Definition 4, it follows that $P \Downarrow V$ for some V, as required for computational adequacy.

Therefore, to complete the proof of Theorem 3 we need to demonstrate that when D is a minimal invariant for $(\mathbb{Z} + (-) \to (-))_\bot$, there exists a relation $\lhd$ as in Definition 4.

3 A new construction of the relation $\lhd$

Let us begin by pointing out why the existence of a relation $\lhd$ as in Definition 4 is problematic. One can formulate the problem as one of solving a certain fixed point equation. Let $\mathcal{R}el$ be the set of all binary relations $R \subseteq D \times Prog$ which contain $\{\bot\} \times Prog$ and which satisfy condition 2 of Definition 4. In other words a binary relation R is in $\mathcal{R}el$ if and only if for each $P \in Prog$, $\{d \mid (d, P) \in R\}$ is an *admissible* subset of the domain D, i.e. chain-complete and containing $\bot$. Define an operation $\phi : \mathcal{R}el \longrightarrow \mathcal{R}el$ by:

$$\begin{aligned}\phi(R) \stackrel{def}{=} \{(d,P) \mid\ & d = \bot \vee \exists n(d = \mathrm{num}(n) \wedge P \Downarrow \underline{n}) \vee \\ & \exists f, \lambda x.M\,(d = \mathrm{fun}(f) \wedge P \Downarrow \lambda x.M \wedge \\ & \qquad \forall (d', P') \in R.(f(d'), M[P'/x]) \in R)\}\ .\end{aligned}$$

Then a formal approximation relation is precisely an element $\lhd \in \mathcal{R}el$ satisfying $\lhd = \phi(\lhd)$. It is easy to see that $\mathcal{R}el$ is closed under taking intersections of binary relations, and hence it is a complete lattice when ordered by inclusion, $\subseteq$. However, ϕ is not a monotonic operation for $\subseteq$ (since the definition of $\phi(R)$ contains a negative as well as a positive occurrence of R), so we cannot appeal to the familiar Tarski fixed point theorem to construct a fixed point for ϕ.

In the literature, two methods can be found for constructing relations on recursively defined domains with certain non-monotonic fixed point properties. One method, due to Milne, Plotkin and Reynolds, makes use of Scott's construction of a recursively defined domain $D \cong \Phi(D)$ as the colimit of a chain of embedding-projections $D_0 \longrightarrow D_1 \longrightarrow \cdots$, where the domain D_n is obtained by iterating the domain constructor $\Phi(-)$ n times, starting with the trivial domain $\{\bot\}$. Then $\lhd$ can be constructed as an inverse limit of relations $\lhd_n \subseteq D_n \times Prog$ built up by iterating an appropriate action of $\Phi(-)$ on relations; see [12].

A second method, essentially due to Martin-Löf, applies only to Scott domains (precluding the use of constructors like the Plotkin powerdomain) and makes use of their presentation in terms of 'information systems' [13]. This method hinges upon the fact that for each program P, $\{d \mid d \lhd P\}$ is in fact a Scott-closed subset of D. Hence it suffices to construct the relation $\lhd$ only for compact elements of D, since $d \lhd P$ holds if and only if $a \lhd P$ holds for all compact a with $a \sqsubseteq d$. Information systems provide a formal language for compact elements of (recursively defined) Scott domains, and $a \lhd P$ (a compact) can be defined by a well-founded induction on the size of (a formal representation of) a. See [15, Sect. 13.4].

Here we present a third method, which is more abstract than the above two in that it relies upon the 'minimal invariant' property of Definition 2 rather than either of the techniques for giving concrete constructions of recursively defined domains mentioned above. To begin with, following Freyd's recent work on recursive types [1, 2, 3], we separate the positive and negative occurrences of R in the definition of $\phi(R)$. Thus given two relations $R^-, R^+ \in \mathcal{R}el$, define:

$$\psi(R^-, R^+) \stackrel{def}{=} \{(d, P) \mid d = \bot \vee \exists n(d = \mathrm{num}(n) \wedge P \Downarrow \underline{n}) \vee \exists f, \lambda x.M\,(d = \mathrm{fun}(f) \wedge P \Downarrow \lambda x.M \wedge \forall (d', P') \in R^-.(f(d'), M[P'/x]) \in R^+)\} \ .$$

Clearly ψ determines a monotonic function

$$\psi : \mathcal{R}el^{op} \times \mathcal{R}el \longrightarrow \mathcal{R}el$$

where $\mathcal{R}el$ is partially ordered via $\subseteq$ and where $\mathcal{R}el^{op}$ has the opposite ordering. Furthermore, ϕ can be recovered from ψ by diagonalizing:

$$\phi(R) = \psi(R, R) \ . \tag{3}$$

We remarked above that $\mathcal{R}el$ is a complete lattice, with infima given by set-theoretic intersection. Hence $\mathcal{R}el^{op} \times \mathcal{R}el$ is also a complete lattice. We obtain a monotonic operator

$$\psi^{\S} : \mathcal{R}el^{op} \times \mathcal{R}el \longrightarrow \mathcal{R}el^{op} \times \mathcal{R}el$$

on this complete lattice by 'symmetrizing' ψ:

$$\psi^{\S}(R^-, R^+) \stackrel{def}{=} (\psi(R^+, R^-), \psi(R^-, R^+)) \ .$$

Now we can apply Tarski's fixed point theorem to obtain the least fixed point of $\psi^{\S}$, which we will denote by $(\lhd^-, \lhd^+)$. Thus $\lhd^-$ and $\lhd^+$ are given by simultaneous, inductive definitions. Using the fact that infima in $\mathcal{R}el$ are given by intersection, together with the definition of $\psi^{\S}$, these relations can be described explicitly as follows:

$$\lhd^+ \stackrel{def}{=} \bigcap\{R^+ \in \mathcal{R}el \mid \exists R^- \in \mathcal{R}el(R^- \subseteq \psi(R^+, R^-) \wedge \psi(R^-, R^+) \subseteq R^+)\}$$

$$\lhd^- \stackrel{def}{=} \bigcap\{S \in \mathcal{R}el \mid \forall R^-, R^+ \in \mathcal{R}el. (R^- \subseteq \psi(R^+, R^-) \wedge \psi(R^-, R^+) \subseteq R^+ \Rightarrow R^- \subseteq S)\} \ .$$

All we need to know about $(\lhd^-, \lhd^+)$ is that it is the least *pre-fixed* point of $\psi^{\S}$. Writing out this least pre-fixed point property for $\psi^{\S}$ on $\mathcal{R}el^{op} \times \mathcal{R}el$ purely in terms of ψ and $\mathcal{R}el$, we obtain the following characteristic properties of $\lhd^-, \lhd^+$ which have a mixed inductive/co-inductive flavour.

Lemma 5. *1.* $\lhd^- = \psi(\lhd^+, \lhd^-)$ *and* $\psi(\lhd^-, \lhd^+) = \lhd^+$.
2. For all $R^-, R^+ \in \mathcal{R}el$, *if*

$$R^- \subseteq \psi(R^+, R^-) \qquad \textit{and} \qquad \psi(R^-, R^+) \subseteq R^+$$

then

$$R^- \subseteq \lhd^- \qquad \textit{and} \qquad \lhd^+ \subseteq R^+ \ .$$

Theorem 6 (Existence of $\lhd$). *When D is a minimal invariant for the domain constructor $(\mathbb{Z} + (-) \to (-))_\perp$, the relations $\lhd^-$ and $\lhd^+$ are equal, and yield a formal approximation relation as in Definition 4.*

Proof. First note that by (3) and part 1 of Lemma 5, if $\lhd^- = \lhd^+$ then this relation is a fixed point for the operation ϕ and hence has the properties required by Definition 4.

We split the equality $\lhd^- = \lhd^+$ into two inclusions. The inclusion $\lhd^+ \subseteq \lhd^-$ follows immediately from Lemma 5, since by clause 1 we may take $R^- = \lhd^+$ and $R_+ = \lhd^-$ in clause 2. So it remains to prove that $\lhd^- \subseteq \lhd^+$.

It is only at this point that we need the minimal invariant property of D. Recall that it says that id_D is the least fixed point of the continuous function $\delta_\Phi : (D \to D) \longrightarrow (D \to D)$ defined in (2). We introduce the following piece of notation: given $R, S \in \mathcal{R}el$ and a continuous function $e \in (D \to D)$, write

$$e : R \subset S$$

to mean that for all $(d, P) \in R$, $(e(d), P) \in S$. From the definition of δ_Φ, it is straightforward to verify that

$$e : R \subset S \Rightarrow \delta_\Phi(e) : \psi(S, R) \subset \psi(R, S) \ .$$

So taking $R = \lhd^-$ and $S = \lhd^+$ and using part 1 of Lemma 5, we have that δ_Φ maps the set

$$\{e \in D \to D \mid e : \lhd^- \subset \lhd^+\} \tag{4}$$

into itself. Clearly this subset of $D \to D$ is chain-closed and contains $\perp$, because of the admissibility condition elements of $\mathcal{R}el$ satisfy. Hence by the familiar fixed point induction principle (see [15, Sect. 10.2] for example), id_D, being the least fixed point of δ_Φ, lies in the subset (4). Thus $id_D : \lhd^- \subset \lhd^+$, which is just to say that $\lhd^- \subseteq \lhd^+$. □

4 Further development

The method of construction of $\lhd$ we have given in this paper can be used quite generally to construct recursively specified relations on recursively defined domains without having to delve into the details of the construction of the domain. Moreover, the construction applies to many different notions of 'relation' on a domain. (Here for example, a relation on D has meant a subset of $D \times Prog$.) The construction can be phrased in terms of an abstract notion of 'relational structure' on a category of domains and of the 'action' of domain constructors on relations, due to O'Hearn and Tennent [7]. This general form of the construction is described in [9, Sect. 5]. That paper treats the case of *unary* relational structures, but the method generalizes easily to n-ary relations. For example, we believe that the recursively specified relation between two recursively defined domains employed by Reynolds to relate a direct and a continuation semantics

of an untyped functional language in [12] can be constructed by applying our method to a suitable binary relational structure.

As pointed out in [9], the method of construction not only provides a simpler construction of certain relations, but also *characterizes* these relations uniquely via a 'universal property'. For instance, by virtue of Lemma 5 (and the fact that $\lhd^- = \lhd = \lhd^+$), the formal approximation relation $\lhd$ is a 'mixed' fixed point in the sense of the following definition.

Definition 7 (Mixed fixed point). Let $(\mathcal{R}, \leq)$ be a partially ordered set and let $\psi : \mathcal{R}^{\mathrm{op}} \times \mathcal{R} \longrightarrow \mathcal{R}$ be a monotonic function. Then $M \in \mathcal{R}$ is a *mixed fixed point* for ψ if

$$M = \psi(M, M) \tag{5}$$

and

$$\forall R, S \in \mathcal{R}(R \leq \psi(S, R) \wedge \psi(R, S) \leq S \Rightarrow R \leq M \leq S) \ . \tag{6}$$

Note that the mixed fixed point of ψ is unique if it exists. Indeed, if $R \in \mathcal{R}$ satisfies $R = \psi(R, R)$, then (6) implies that $R \leq M \leq R$, i.e. $R = M$.

It is not hard to see that conditions (5) and (6) are equivalent to saying that (M, M) is the least pre-fixed point of the monotonic operator

$$(R, S) \mapsto (\psi(S, R), \psi(R, S))$$

on $\mathcal{R}^{\mathrm{op}} \times \mathcal{R}$; or to saying that (M, M) is the greatest post-fixed point of that operator. In fact Definition 7 is the special case for monotonic functions of the condition on functors of mixed variance formulated by Freyd in his work on 'algebraically compact' categories [2, 3]. One can summarize the results in [9, Sect. 5] as establishing that the algebraic compactness property of the category of domains and strict continuous functions is inherited by categories of 'domains equipped with relations' (for a very general notion of relation). As the rest of that paper demonstrates, from the mixed fixed point property of recursively defined relations it is possible to derive a number of induction and co-induction [8] principles for reasoning about the properties of recursively defined domains.

References

1. P. J. Freyd. Recursive Types Reduced to Inductive Types. In *Proc. 5th Annual Symp. on Logic in Computer Science, Philadelphia, 1990* (IEEE Computer Society Press, Washington, 1990), pp 498–508.
2. P. J. Freyd. Algebraically Complete Categories. In A. Carboni *et al* (eds), *Proc. 1990 Como Category Theory Conference*, Lecture Notes in Math. Vol. 1488 (Springer-Verlag, Berlin, 1991), pp 95–104.
3. P. J. Freyd. Remarks on algebraically compact categories. In M. P. Fourman, P. T. Johnstone and A. M. Pitts (eds), *Applications of Categories in Computer Science*, L.M.S. Lecture Note Series 177 (Cambridge University Press, 1992), pp 95–106.
4. C. A. Gunter. *Semantics of Programming Languages. Structures and Techniques.* (MIT Press, 1992.)

5. A. R. Meyer. Semantical Paradigms: Notes for an Invited Lecture. In *Proc. 3rd Annual Symp. on Logic in Computer Science, Edinburgh, 1988* (IEEE Computer Society Press, Washington, 1988), pp 236–255.
6. R. E. Milne. *The formal semantics of computer languages and their implementations*, Ph.D. Thesis, Univ. Cambridge, 1973.
7. P. W. O'Hearn and R. D. Tennent. Relational Parametricity and Local Variables. In *Conf. Record 20th Symp. on Principles of Programming Languages, Charleston, 1993* (ACM, New York, 1993), pp 171–184.
8. A. M. Pitts. A Co-induction Principle for Recursively Defined Domains, *Theoretical Computer Science*, to appear. (Available as Univ. Cambridge Computer Laboratory Tech. Rept. No. 252, April 1992.)
9. A. M. Pitts. Relational Properties of Recursively Defined Domains. In: *Proc. 8th Annual Symp. on Logic in Computer Science, Montréal, 1993* (IEEE Computer Soc. Press, Washington, 1993), pp 86–97.
10. G. D. Plotkin. LCF Considered as a Programming Language, *Theoretical Computer Science* 5(1977) 223–255.
11. G. D. Plotkin. Lectures on Predomains and Partial Functions. Notes for a course at CSLI, Stanford University, 1985.
12. J. C. Reynolds. On the Relation between Direct and Continuation Semantics. In J. Loeckx (ed.), *2nd Int. Colloq. on Automata, Languages and Programming*, Lecture Notes in Computer Science, Vol. 14 (Springer-Verlag, Berlin, 1974), pp 141–156.
13. D. S. Scott. Domains for Denotational Semantics. In M. Nielsen and E. M. Schmidt (eds), *Proc. 9th Int. Coll. on Automata, Languages and Programming*, Lecture Notes in Computer Science, Vol. 140 (Springer, Berlin, 1982), pp 577–613.
14. M. B. Smyth and G. D. Plotkin. The Category-Theoretic Solution of Recursive Domain Equations, *SIAM J. Computing* 11(1982) 761–783.
15. G. Winskel. *The Formal Semantics of Programming Languages. An Introduction.* (MIT Press, 1993.)

This article was processed using the LaTeX macro package with LLNCS style