A Co-induction Principle for Recursively Defined Domains[1]

Andrew M. Pitts

University of Cambridge Computer Laboratory

New Museums Site, Pembroke Street

Cambridge CB2 3QG, England

⟨Andrew.Pitts@cl.cam.ac.uk⟩

[1] To appear in *Theoretical Computer Science*

Abstract

This paper establishes a new property of predomains recursively defined using the cartesian product, disjoint union, partial function space and convex powerdomain constructors. We prove that the partial order on such a recursive predomain D is the greatest fixed point of a certain monotone operator associated to D. This provides a structurally defined family of proof principles for these recursive predomains: to show that one element of D approximates another, it suffices to find a binary relation containing the two elements that is a post-fixed point for the associated monotone operator. The statement of the proof principles is independent of any of the various methods available for explicit construction of recursive predomains. Following Milner and Tofte [10], the method of proof is called *co-induction*. It closely resembles the way bisimulations are used in concurrent process calculi [9].

Two specific instances of the co-induction principle already occur in work of Abramsky [2, 1] in the form of 'internal full abstraction' theorems for denotational semantics of SCCS and the lazy lambda calculus. In the first case post-fixed binary relations are precisely Abramsky's *partial bisimulations*, whereas in the second case they are his *applicative bisimulations*. The co-induction principle also provides an apparently useful tool for reasoning about equality of elements of recursively defined datatypes in (strict or lazy) higher order functional programming languages.

1 Introduction

Recursively defined domains play a key role in giving denotational semantics for many programming language features. In particular, such domains arise naturally in connection with the recursive datatypes of higher order functional programming languages such as Standard ML [11] or Haskell [7]. Elegant methods for constructing recursive domains have been devised—such as via Scott's 'information systems' (see [21]). Nevertheless, the structure of recursive domains can be very complicated, especially for domain equations involving the (partial) function space constructor. Ideally one would like to have proof principles which permit reasoning about recursive domains without recourse to an explicit description of their structure.

This paper introduces such a proof principle, which applies uniformly to all predomains recursively defined using the cartesian product, disjoint union, partial function space and convex powerdomain constructors. We show that the partial order on such a recursive predomain D is the greatest fixed point of a certain monotone operator associated to D. This provides a structurally defined family of proof principles for these recursive predomains: to show that one element of D approximates another, it suffices to find a binary relation containing the two elements that is a post-fixed point for the associated monotone operator. The statement of the proof principles is independent of any of the various methods available for explicit construction of recursive predomains (using colimits of embedding-projection pairs [20], or using information systems [21], for example). Following Milner and Tofte [10], the method of proof is called *co-induction*. It closely resembles the way bisimulations are used in concurrent process calculi [9].

Two specific instances of the co-induction principle already occur in work of Abramsky [2, 1], in the form of 'internal full abstraction' theorems for denotational semantics of SCCS and the lazy lambda calculus. In the first case post-fixed binary relations are precisely Abramsky's *partial bisimulations*, whereas in the second case they are his *applicative bisimulations* (see also Howe [6]). The co-induction principle also provides an apparently useful tool for reasoning about equality of elements of recursively defined datatypes in (strict or lazy) higher order functional programming languages.

In the first part of the paper we restrict attention to the simple case of a single domain equation involving products, disjoint unions, lifting and partial function spaces. Simultaneous and parameterized domain equations are considered in section 4. The co-induction property for recursive domains involving the convex (or Plotkin) powerdomain is established in section 5. Finally, in section 6 we show how to extend the co-induction principle to cope with the combination of polymorphism and recursive datatype declarations to be found in (the functional fragment of) Standard ML.

Throughout, we will work with *pre*domains rather than with domains. In other words, the existence of a least element in a semantic domain is not assumed. Plotkin [19] uses partial continuous functions between predomains. Here we will use total continuous functions to lifted predomains. This necessarily places an emphasis upon the role of the lifting construct (adjoining a least element to a predomain). Although this has some drawbacks from the point of notational complexity, it emphasises an important conceptual distinction in the semantics of datatypes, namely that between *values* (or canonical expressions) and *computations* of those values. Consider for example, the Standard ML datatype declaration:

```
datatype      nat  =  Zero | Suc of nat
and       natlist  =  Nil | Cons of nat * natlist
```

Since Standard ML is a strict language, the *values* of type `natlist` are intended to comprise the set $\mathbf{N}^*$ of lists of natural numbers. A denotational semantics for datatypes using predomains can respect this intention, since sets are particular kinds of predomain—the *discrete* ones, in which the partial order coincides with equality. General expressions of type `natlist` (in a pure functional subset of ML, say) do not denote values, but rather computations of values in `natlist` and are assigned denotations in the lifted predomain $(\mathbf{N}^*)_\perp$. (A similar separation of values from computations can be made for the denotational semantics of lazy functional languages using predomains and lifting, although of course the predomain for `natlist` would no longer be discrete.)

As [19] shows, for languages involving higher order functions this distinction between computations and values in their denotational semantics is important to achieve a good fit between operational and denotational semantics at higher types. It is also a key conceptual aspect of Moggi's [12, 13] modular approach to denotational semantics using categorical monads. An interesting question that we leave for future work is to what extent the lifting monad can be replaced by other 'computational' monads in order to extend the co-induction principle described here to cope with datatypes in 'impure' languages such as Standard ML which mix imperative constructs with higher order functional programming.

2 Simulations

We begin by fixing the notation for predomains that will be use here. When we come to consider convex powerdomains in section 5 we will need to be more restrictive, but for the present, by a *predomain* we just mean a set D equipped with a partial order $\sqsubseteq_D$ and possessing least upper bounds of all countable $\sqsubseteq_D$-chains. Let $\mathcal{D}$ denote the collection of all predomains. We consider the following constructions on $D, E \in \mathcal{D}$.

Cartesian product of D and E is the set of ordered pairs $D \times E = \{(x, y) \mid x \in D \wedge y \in E\}$, partially ordered componentwise: $(x, y) \sqsubseteq_{D\times E} (x', y')$ if and only if $x \sqsubseteq_D x'$ and $y \sqsubseteq_E y'$.

Disjoint union of D and E has underlying set $D + E = \{\mathrm{inl}(x) \mid x \in D\} \cup \{\mathrm{inr}(y) \mid y \in E\}$, where $x \mapsto \mathrm{inl}(x)$ and $y \mapsto \mathrm{inr}(y)$ are injective functions with disjoint images. The partial order on $D+E$ is: $z \sqsubseteq_{D+E} z'$ if and only if either $z = \mathrm{inl}(x)$ and $z' = \mathrm{inl}(x')$ for some $x, x' \in D$ with $x \sqsubseteq_D x'$, or $z = \mathrm{inr}(y)$ and $z' = \mathrm{inl}(y')$ for some $y, y' \in E$ with $y \sqsubseteq_E y'$.

Lift of D has underlying set $D_\perp = \{[x] \mid x \in D\} \cup \{\perp\}$, where $x \mapsto [x]$ is an injective function whose image does not contain the element $\perp$. The partial order on $D_\perp$ is: $u \sqsubseteq_{D_\perp} u'$ if and only if for all $x \in D$, if $u = [x]$ then $u' = [x']$ for some $x' \in D$ with $x \sqsubseteq_D x'$.

Exponential of E by D has underlying set $D \rightarrow E$ containing all functions from D to E that preserve least upper bounds of countable chains. These function are partially ordered pointwise from E: $f \sqsubseteq_{D \rightarrow E} f'$ if and only if for all $x \in D$, $f(x) \sqsubseteq_E f'(x)$.

Partial exponential of E by D is defined to be the exponential of $E_\perp$ by D, and is denoted $D \rightharpoonup E$.

Definition 2.1 By a *predomain constructor* we mean a formal expression σ given by the following grammar.

$$
\begin{array}{rcll}
\sigma & ::= & K & \text{constant} \\
& | & \alpha & \text{variable} \\
& | & \sigma \times \sigma & \text{product} \\
& | & \sigma + \sigma & \text{disjoint union} \\
& | & \sigma_\perp & \text{lift} \\
& | & \sigma \rightharpoonup \sigma & \text{partial exponential}
\end{array}
$$

where K ranges over $\mathcal{D}$ and α stands for a variable member of $\mathcal{D}$. For the moment we only consider domain constructors involving a single variable. Given such a σ and $D \in \mathcal{D}$, then $\sigma(D)$ will denote the predomain resulting from replacing α by D in σ and interpreting the constructors $\times, +, {}_\perp, \rightharpoonup$ as above.

The process by which $\sqsubseteq_{D_\perp}$ is obtained from $\sqsubseteq_D$ is a particular case of the following operation sending binary relations $\lhd$ on a predomain D to binary relations $\lhd_\perp$ on the lifted predomain $D_\perp$. We define $\lhd_\perp$ by requiring $u \lhd_\perp u'$ to hold if and only if for all $x \in D$, if $u = [x]$ then $u' = [x']$ for some $x' \in D$ with $x \lhd x'$.

Using this operation of lifting binary relations, we can give a similar operation for each predomain constructor.

Definition 2.2 Given $D \in \mathcal{D}$ and a binary relation $\lhd$ on $D_\perp$, for each predomain constructor σ define a binary relation $\sigma(\lhd)$ on $\sigma(D)_\perp$ by induction on the structure of σ, as follows:

Case σ is K: $\sigma(\lhd)$ is $\sqsubseteq_{K_\perp}$.

Case σ is α: $\sigma(\lhd)$ is $\lhd$.

Case σ is $\sigma_1 \times \sigma_2$: $u\ \sigma(\lhd)\ u'$ if and only if

> for all $(x_1, x_2) \in \sigma_1(D) \times \sigma_2(D)$, if $u = [(x_1, x_2)]$ then $u' = [(x'_1, x'_2)]$ for some $(x'_1, x'_2) \in \sigma_1(D) \times \sigma_2(D)$ with $[x_1]\ \sigma_1(\lhd)\ [x'_1]$ and $[x_2]\ \sigma_2(\lhd)\ [x'_2]$.

Case σ is $\sigma_1 + \sigma_2$: $u\ \sigma(\lhd)\ u'$ if and only if

- for all $x_1 \in \sigma_1(D)$, if $u = [\mathrm{inl}(x_1)]$ then $u' = [\mathrm{inl}(x'_1)]$ for some $x'_1 \in \sigma_1(D)$ with $[x_1]\ \sigma_1(\lhd)\ [x'_1]$; and
- for all $x_2 \in \sigma_2(D)$, if $u = [\mathrm{inr}(x_2)]$ then $u' = [\mathrm{inr}(x'_2)]$ for some $x'_2 \in \sigma_2(D)$ with $[x_2]\ \sigma_2(\lhd)\ [x'_2]$.

Case σ is $(\sigma_1)_\perp$: $\sigma(\lhd)$ is the lifted relation $(\sigma_1(\lhd))_\perp$.

Case σ is $\sigma_1 \rightharpoonup \sigma_2$: $u\ \sigma(\lhd)\ u'$ if and only if

> for all $f \in \sigma_1(D) \rightharpoonup \sigma_2(D)$, if $u = [f]$ then $u' = [f']$ for some $f' \in \sigma_1(D) \rightharpoonup \sigma_2(D)$ with $f(x_1)\ \sigma_2(\lhd)\ f'(x_1)$ for all $x_1 \in \sigma_1(D)$.

Recall that an isomorphism between predomains can be specified by a bijection between the underlying sets that both preserves and reflects the partial orders. We write $D \cong E$ when D and E are isomorphic, and write $k : D \cong E$ if k is a particular isomorphism witnessing this fact.

A predomain $D \in \mathcal{D}$ is an invariant for a predomain constructor if $D \cong \sigma(D)$. Since isomorphisms map least elements to least elements when they exist, clearly D is an invariant if and only if $D_\perp \cong \sigma(D)_\perp$. Since in what follows we need only deal with the isomorphism between the lifted predomains, we will say that an *invariant for a predomain constructor* σ is a pair (D, k) with $D \in \mathcal{D}$ and $k : D_\perp \cong \sigma(D)_\perp$. Standard results of domain theory (which we review in the next section) guarantee that all of the predomain constructors of Definition 2.1 possess invariants. For those more familiar with solving domain equations rather than predomain equations, it is perhaps worth pointing out why Definition 2.1 does not contain a clause for exponentials, $\sigma{\to}\sigma'$: with unrestricted use of exponentials, invariants may fail to exist. For example, there is no predomain D satisfying $D \cong D{\to}2$, where 2 is a two-element set regarded as a discrete predomain. (For such a D must be discrete, since it is isomorphic to $D{\to}2$ which is always discrete for any D; so D is a set, $D{\to}2$ is its powerset, and they are in bijection—an impossibility, by the usual Cantor diagonal argument.)

The following is the key notion of this paper.

Definition 2.3 Let (D, k) be an invariant for σ. A *σ-simulation*, $\lhd$, is a binary relation on $D_\perp$ such that for all $u, u' \in D_\perp$, if $u \lhd u'$ then $k(u)\ \sigma(\lhd)\ k(u')$.

Lemma 2.4 *For any $D \in \mathcal{D}$, the binary relation $\sigma(\sqsubseteq_{D_\perp})$ is equal to $\sqsubseteq_{\sigma(D)_\perp}$. Hence the partial order relation $\sqsubseteq_{D_\perp}$ is always a σ-simulation for any invariant (D, k).*

Proof The first sentence follows from Definition 2.2 by induction on the structure of σ. The second sentence follows from the first, because k is order preserving. □

Let $\mathcal{R}e\ell(D)$ denote the set of binary relations on the underlying set of $D \in \mathcal{D}$, partially ordered by set-theoretic inclusion. The mapping $\lhd \mapsto \sigma(\lhd)$ of Definition 2.2 defines a monotone function from $\mathcal{R}e\ell(D_\perp)$ to $\mathcal{R}e\ell(\sigma(D)_\perp)$. Monotonicity can be proved by induction on the structure of σ. The interesting case is when σ is a partial exponential, $\sigma_1 \rightharpoonup \sigma_2$ say: since the relevant clause of Definition 2.2 has a positive occurrence of $\sigma_2(\lhd)$ and no occurrence of $\sigma_1(\lhd)$, monotonicity of the operation on binary relations is preserved. Thus given an invariant (D, k) for σ, the function

$$\lhd \quad \mapsto \quad \{(u, u') \mid k(u)\ \sigma(\lhd)\ k(u')\} \tag{1}$$

determines a monotone operator on $\mathcal{R}e\ell(D_\perp)$. Let us write $\preceq$ for the greatest fixed point of this operator. Since σ-simulations for (D, k) are precisely the post-fixed points for this monotone operator, Lemma 2.4 implies that $\sqsubseteq_{D_\perp}$ is contained in $\preceq$. We will call the invariant (D, k) *extensional* if the reverse inclusion holds, so that $\sqsubseteq_{D_\perp}$ is the greatest fixed point for (1).

The main technical result of this paper is that recursively defined predomains are extensional in the above sense. We will write $\mathrm{rec}\alpha.\sigma$ for the predomain recursively defined by the equation $\alpha = \sigma$. This predomain comes equipped with an isomorphism k that not only makes $(\mathrm{rec}\alpha.\sigma, k)$ an invariant for σ, but which is the minimal such invariant in a suitable sense (such as being initial for embedding-projection pairs). In section 3 we will recall enough of the construction and characteristic properties of such recursive predomains to prove:

Theorem 2.5 (Co-induction property) *Let σ be a predomain constructor as in Definition 2.1, and let $D = \mathrm{rec}\alpha.\sigma$ be the predomain recursively defined by the equation $\alpha = \sigma$. For any $u, u' \in D_\perp$, $u \sqsubseteq_{D_\perp} u'$ holds if and only if there is some σ-simulation $\lhd$ with $u \lhd u'$.*

The force of the theorem is to provide a method for proving that one element in (the lift of) a recursively defined predomain approximates another: one just has to find a simulation relating the two elements. It is clear from the definition of simulation that this property of recursive domains could be formalized in a second order logic such as the μ-calculus of Scott, de Bakker and Park [14]. Thus the method can be used independently of any particular explicit description of recursive domains within higher order logic (such as that used in the next section to prove the theorem). Note that in contrast to domain-theoretic induction principles (such as Scott's fixed point induction), the co-induction principle does not require us to restrict attention to chain-complete relations on a domain. In practice we may not need to construct a very large simulation in order to establish an instance of the partial order relation. Here is an example involving lazy lists.

Example 2.6 (Lazy lists) The Standard ML [11] datatype of (head-strict) lazy integer lists (or 'sequences' [15, 5.12]) is

```
datatype seq = Nil | Cons of int*(unit -> seq)
```

Consider the following ML declarations defining expressions `sucq`, `from` and `natq` of types `seq->seq`, `int->seq` and `unit->seq` respectively:

```
fun  sucq Nil = Nil
  |  sucq (Cons(n,s)) = Cons(n+1 , fn()=>sucq(s()) );

fun  from n = Cons(n , fn()=>from(n+1) );

fun  natq() = Cons(0 , fn()=>sucq(natq()));
```

We will use Theorem 2.5 to prove that `natq()` and `from(0)` have equal denotations.

The datatype `seq` can be given a denotational semantics using the initial solution I of the predomain equation $\alpha = 1 + \mathbb{Z} \times \alpha_\perp$, with $1 = \{0\}$ a one-element set and $\mathbb{Z} = \{\ldots, -1, 0, 1, \ldots\}$ the set of integers (both sets being regarded as predomains via the discrete partial ordering). Closed ML expressions of type `seq` receive denotations in the domain $I_\perp$.

Let us introduce the following notation in connection with the predomain I. Write nil and $n :: u$ for the elements of I corresponding under the canonical isomorphism $i : I \cong 1 + \mathbb{Z} \times I_\perp$ to $\mathrm{inl}(0)$ and $\mathrm{inr}(n, u)$ respectively. Let $k : I_\perp \cong (1 + \mathbb{Z} \times I_\perp)_\perp$ denote the lifted isomorphism, $i_\perp$. For $u \in I_\perp$ and $x \in I$, write $u \Downarrow x$ if $u = [x]$.

Then from Definition 2.2 we have that a $(1 + \mathbb{Z} \times \alpha_\perp)$-simulation for the invariant (I, k) is a binary relation $\lhd$ on $I_\perp$ such that for all $u, u', v \in I_\perp$ and all $n \in \mathbb{Z}$

(a) if $u \lhd u'$ and $u \Downarrow nil$, then $u' \Downarrow nil$; and

(b) if $u \lhd u'$ and $u \Downarrow n :: v$, then $u' \Downarrow n :: v'$ for some v' with $v \lhd v'$.

The denotation of `sucq` is of the form $[s] \in (I \rightharpoonup I)_\perp$, where s is a (continuous) function $I \to I_\perp$ satisfying

$$\begin{aligned} s(nil) &= [nil] \\ s(n :: u) &= [(n+1) :: s^*(u)] \end{aligned}$$

with s^* the strict function from $I_\perp$ to $I_\perp$ corresponding to s, i.e.

$$s^*(u) = \begin{cases} s(x) & \text{if } u = [x], \text{ some } x \in I \\ \perp & \text{if } u = \perp \end{cases}$$

The denotation of `from` is of the form $[f] \in (\mathbb{Z} \rightharpoonup I)_\perp$, where f is a (continuous) function $\mathbb{Z} \to I_\perp$ satisfying

$$f(n) = [n :: f(n+1)] \tag{2}$$

Then the denotation of `from(0)` is $f(0) \in I_\perp$, whereas the denotation of `natq()` is some element $u \in I_\perp$ satisfying

$$u = [0 :: s^*(u)]$$

So we have to prove that $u = f(0)$.

By Theorem 2.5, it suffices to exhibit binary relations $\lhd, \lhd'$ on $I_\perp$ satisfying conditions (a) and (b), and such that $u \lhd f(0)$ and $f(0) \lhd' u$. But it is not hard to see that these requirements are met by

$$\begin{array}{rcl} \lhd & = & \{((s^*)^n(u), f(n)) \mid n \geq 0\} \\ \lhd' & = & \{(f(n), (s^*)^n(u)) \mid n \geq 0\} \end{array}$$

where $(s^*)^n$ denotes the function s^* applied n times. Since $(s^*)^0(u) = u$, we certainly have $u \lhd f(0)$ and $f(0) \lhd' u$. To verify conditions (a) and (b), first note that by induction on $n \geq 0$

$$(s^*)^n(u) \quad = \quad [n :: (s^*)^{n+1}(u)] \qquad (3)$$

By (2) and (3), for no $n \geq 0$ do we have $(s^*)^n(u) \Downarrow \mathit{nil}$ or $f(n) \Downarrow \mathit{nil}$. So $\lhd$ and $\lhd'$ trivially satisfy (a). For (b), given the instance $((s^*)^n(u), f(n))$ of $\lhd$, if $(s^*)^n(u) \Downarrow m :: v$, then by (3) $m = n$ and $v = (s^*)^{n+1}(u)$. So taking $v' = f(n+1)$, by (2) we have $f(n) \Downarrow m :: v'$ with $v \lhd v'$, as required. Similarly, (b) for $\lhd'$ follows from (2) and (3) by a symmetric argument.

Examining the above argument, it seems that we have proved $u = f(0)$ for an *arbitrary* extensional invariant for $1 + \mathbb{Z} \times \alpha_\perp$, and not just for the initial one I. In fact for this predomain constructor it is the case that the co-induction property of Theorem 2.5 suffices to characterize the initial solution uniquely up to isomorphism amongst all invariants. We will return to this point in section 7.

It is instructive to compare the proof in Example 2.6 with proofs exploiting specific properties of the datatype `seq`. For example, Bird and Wadler [4, page 184] use the so-called 'take-lemma' to form finite approximations to lazy lists, and hence reduce the proof to a suitable induction over the natural numbers. As we recall in the next section, a general recursively defined domain comes with a notion of finite approximation, by virtue of its construction as a colimit of a chain of embeddings. We exploit this to reduce the proof of the co-induction property to an application of mathematical induction (see Proposition 3.3). So the co-induction principle is in a sense just a repackaging of an inductive arguement on finite approximations. However, the induction is done once and for all in establishing the general principle, allowing us to avoid consideration of such finite approximations in any particular application of the principle. This is a distinct advantage when the particular notion of finite approximation is complicated—as is generally the case for datatypes involving the function type constructor. Here is such an example.

Example 2.7 (Lazy lambda calculus) Consider the recursive predomain $D = \mathrm{rec}\alpha.\sigma$ with $\sigma = \alpha_\perp \rightharpoonup \alpha$. If k denotes the lift of the canonical isomorphism $D \cong D_\perp \rightharpoonup D$, then k is an isomorphism between the domain $L = D_\perp$ and the domain $(L \to L)_\perp$. L is the canonical model of the lazy untyped lambda calculus studied by Abramsky and Ong [1, 3].

For $u \in L$ and $f \in L \to L$, write $u \Downarrow f$ if $k(u) = [f]$. Then from Definition 2.2 we have that a σ-simulation for (D, k) is a binary relation $\lhd$ on L satisfying

> if $u \lhd u'$ and $u \Downarrow f$, then $u' \Downarrow f'$ for some $f' \in L \to L$ with $f(v) \lhd f'(v)$ for all $v \in L$.

This is the notion of *applicative bisimulation* used in [1]. The co-induction property for this σ is Abramsky's 'internal full abstraction' result ([1, Theorem 4.1]) for the canonical model of the lazy lambda calculus .

Remark 2.8 (σ-bisimulations) Clearly Theorem 2.5 can be applied to prove that an equality holds by splitting the goal of proving $u = u'$ into the two subgoals $u \sqsubseteq u'$ and $u' \sqsubseteq u$. However, it is also possible to give a version of the theorem which directly characterizes the equality relation on $(\mathrm{rec}\alpha.\sigma)_\perp$ as the largest binary relation $\sim$ satisfying

> $u \sim u'$ implies $k(u)\ \sigma\langle\sim\rangle\ k(u')$

where $\sim \mapsto \sigma\langle\sim\rangle$ is obtained from Definition 2.2 by 'symmetrizing' each clause of the definition. For example, the symmetric version of the lift operation on relations sends $\sim$ to the relation $\sim_\perp \cap ((\sim^{\mathrm{op}})_\perp)^{\mathrm{op}}$, i.e. the relation containing all pairs (u, u') satisfying

for all x, if $u = [x]$ then $u' = [x']$ for some x' with $x \sim x'$; and

for all x', if $u' = [x']$ then $u = [x]$ for some x with $x \sim x'$.

3 Embeddings

In this section we will recall from [20] enough of the theory of solving domain equations using colimits of embedding-projection pairs to prove Theorem 2.5. Although *loc. cit.* uses domains rather than predomains, the theory is easily adapted.

As usual, we call a function $i : D \to E$ between predomains *continuous* if it preserves the least upper bound of any ω-chain. Such a continuous function is an *embedding* if there is a continuous function $i^\circ : E \to D_\perp$ satisfying

- for all $x \in D$, $[x] = i^\circ(i(x))$, and
- for all $x \in D$ and $y \in E$, if $[x] \sqsubseteq i^\circ(y)$ then $i(x) \sqsubseteq y$.

It is not hard to see that such an i° is uniquely determined by i. We call i° the *partial projection* associated to the embedding i. We will use the following notation to indicate that i is an embedding:

$$i : D \hookrightarrow E$$

Embeddings compose: given $i : D \hookrightarrow E$ and $j : E \hookrightarrow F$, the function composition $j \circ i : D \to F$ is an embedding with associated partial projection given by

$$(j \circ i)^\circ \;=\; \begin{cases} i^\circ(y) & \text{if } j^\circ(z) = [y], \text{ some } y \in E \\ \perp & \text{if } j^\circ(z) = \perp \end{cases}$$

Clearly the identity function on a predomain D is an embedding, with associated partial projection the insertion $x \mapsto [x]$ of D into $D_\perp$. Thus predomains and embeddings form a category, which we denote by $\mathcal{D}^e$.

The operations on predomain considered in section 2 extend to ones on embeddings between predomains, as follows.

Definition 3.1 Given an embedding $i : D \hookrightarrow E$, for each predomain constructor σ as in Definition 2.1, define an embedding $\sigma(i) : \sigma(D) \hookrightarrow \sigma(E)$ by induction on the structure of σ, as follows:

Case σ is K: $\sigma(i)$ is the identity function on K. The associated partial projection is $[\_] : K \to K_\perp$.

Case σ is α: $\sigma(i)$ is i itself.

Case σ is $\sigma_1 \times \sigma_2$: given $(x_1, x_2) \in \sigma_1(D) \times \sigma_2(D)$, define

$$\sigma(i)(x_1, x_2) \quad = \quad (\sigma_1(x_1), \sigma_2(x_2))$$

The associated partial projection sends $(y_1, y_2) \in \sigma_1(E) \times \sigma_2(E)$ to

$$\sigma(i)^\circ(y_1, y_2) \quad = \quad \begin{cases} [(x_1, x_2)] & \text{if } \sigma_j(i)^\circ(y_j) = [x_j] \text{ for some } x_j \ (j = 1, 2) \\ \perp & \text{otherwise} \end{cases}$$

Case σ is $\sigma_1 + \sigma_2$: given $z \in \sigma_1(D) + \sigma_2(D)$, define

$$\sigma(i)(z) \quad = \quad \begin{cases} \mathrm{inl}(\sigma_1(x_1)) & \text{if } z = \mathrm{inl}(x_1) \text{ for some } x_1 \\ \mathrm{inr}(\sigma_2(x_2)) & \text{if } z = \mathrm{inr}(x_2) \text{ for some } x_2 \end{cases}$$

The associated partial projection sends $w \in \sigma_1(E) + \sigma_2(E)$ to

$$\sigma(i)^\circ(w) \quad = \quad \begin{cases} [\mathrm{inl}(x_1)] & \text{if } w = \mathrm{inl}(y_1) \text{ for some } y_1, \text{ and} \\ & \qquad \sigma_1(i)^\circ(y_1) = [x_1] \text{ for some } x_1 \\ [\mathrm{inr}(x_2)] & \text{if } w = \mathrm{inr}(y_2) \text{ for some } y_2, \text{ and} \\ & \qquad \sigma_2(i)^\circ(y_2) = [x_2] \text{ for some } x_2 \\ \perp & \text{otherwise} \end{cases}$$

Case σ **is** $(\sigma_1)_\perp$**:** given $u \in \sigma_1(D)_\perp$, define

$$\sigma(i)(u) \quad = \quad \begin{cases} [\sigma_1(i)(x)] & \text{if } u = [x] \text{ for some } x \\ \perp & \text{if } u = \perp \end{cases}$$

The associated partial projection sends $v \in \sigma_1(E)_\perp$ to

$$\sigma(i)^\circ(v) \quad = \quad \begin{cases} [\sigma_1(i)^\circ(y)] & \text{if } v = [y] \text{ for some } y \\ [\perp] & \text{if } v = \perp \end{cases}$$

Case σ **is** $\sigma_1 \rightharpoonup \sigma_2$**:** given $f \in \sigma_1(D) \rightharpoonup \sigma_2(D)$, define $\sigma(i)(f)$ to be the function $\sigma_1(E) \to \sigma_2(E)_\perp$ sending $y_1 \in \sigma_1(E)$ to

$$\sigma(i)(f)(y_1) \quad = \quad \begin{cases} [\sigma_2(i)(x_2)] & \text{if } \sigma_1(i)^\circ(y_1) = [x_1] \text{ for some } x_1, \text{ and} \\ & \qquad f(x_1) = [x_2], \text{ for some } x_2 \\ \perp & \text{otherwise} \end{cases}$$

The associated partial projection sends $g \in \sigma_1(E) \rightharpoonup \sigma_2(E)$ to $[f]$ where $f \in \sigma_1(D) \rightharpoonup \sigma_2(D)$ is the function sending $x_1 \in \sigma_1(D)$ to

$$f(x_1) \quad = \quad \begin{cases} \sigma_2(i)^\circ(y_2) & \text{if } g(\sigma_1(i)(x_1)) = [y_2] \text{ for some } y_2 \\ \perp & \text{otherwise} \end{cases}$$

The above action of a predomain constructor on embeddings preserves identities and composition. So each σ determines a functor $F = \sigma(\_) : \mathcal{D}^e \to \mathcal{D}^e$. By definition, *the predomain* $\mathrm{rec}\alpha.\sigma$ *recursively defined by the equation* $\alpha = \sigma$, is the initial algebra I_F in $\mathcal{D}^e$ for this functor. Thus I_F comes equipped with an embedding $i_F : F(I_F) \hookrightarrow I_F$ such that for any other embedding of the form $j : F(D) \hookrightarrow D$ there is a unique embedding $\hat{\jmath} : I_F \hookrightarrow D$ making the square of embeddings

$$\begin{tikzcd} F(I_F) \arrow[r, hook, "i_F"] \arrow[d, hook, "F(\hat{\jmath})"'] & I_F \arrow[d, hook, "\hat{\jmath}"] \\ F(D) \arrow[r, hook, "j"'] & D \end{tikzcd}$$

commute. As is well known, this property of i_F forces it to be an isomorphism.

The existence of such an $i_F : F(I_F) \hookrightarrow I_F$ follows from the fact that $\mathcal{D}^e$ has, and each $F = \sigma(\_)$ preserves, colimits of ω-chains, together with the fact that $\mathcal{D}^e$ possesses an initial object (the empty predomain $\emptyset$). These properties can be deduced from the corresponding facts about domains in [20] by observing that $\mathcal{D}^e$ is isomorphic to the category of 'embedding-projection pairs' ([20, Definition 6]) between ω-complete posets with least elements. Indeed, when the lift functor $(\_)_\perp$ is applied to an embedding $i : D \hookrightarrow E$ it yields the embedding half of an embedding-projection pair, the projection part of which is the continuous function $i^* : E_\perp \to D_\perp$ given by:

$$i^*(v) \quad = \quad \begin{cases} i^\circ(y) & \text{if } v = [y] \\ \perp & \text{if } v = \perp \end{cases} \tag{4}$$

and every embedding-projection pair between lifted predomains arises uniquely in this way.

We say that a functor $F : \mathcal{D}^e \to \mathcal{D}^e$ is *ω-continuous* if it preserves colimits of ω-chains. Its initial algebra I_F can be constructed as the colimit in $\mathcal{D}^e$ of the chain of embeddings

$$\emptyset \overset{i_0}{\hookrightarrow} F(\emptyset) \overset{i_1}{\hookrightarrow} \cdots \overset{i_{n-1}}{\hookrightarrow} F^n(\emptyset) \overset{i_n}{\hookrightarrow} F^{n+1}(\emptyset) \overset{i_{n+1}}{\hookrightarrow} \cdots \tag{5}$$

where i_0 is uniquely determined and, inductively, $i_{n+1} = F(i_n)$. The structure morphism for the initial algebra, $i_F : F(I_F) \hookrightarrow I_F$, is constructed as follows. Let

$$(k_n : F^n(\emptyset) \hookrightarrow I_F \mid n \in \mathbb{N})$$

denote the colimiting cone for (5). Since F is ω-continuous,

$$(F(k_n) : F^{n+1}(\emptyset) \hookrightarrow F(I_F) \mid n \in \mathbb{N}) \tag{6}$$

is a colimiting cone for the ω-chain

$$F(\emptyset) \overset{F(i_0)}{\hookrightarrow} F^2(\emptyset) \overset{F(i_1)}{\hookrightarrow} \cdots \overset{F(i_{n-1})}{\hookrightarrow} F^{n+1}(\emptyset) \overset{F(i_n)}{\hookrightarrow} F^{n+2}(\emptyset) \overset{F(i_{n+1})}{\hookrightarrow} \cdots$$

Clearly $(k_{n+1} : F^{n+1}(\emptyset) \hookrightarrow I_F \mid n \in \omega)$ is a cone for this ω-chain. Then $i_F : F(I_F) \hookrightarrow I_F$ is the unique factorization of this cone through the colimiting cone (6). Thus for each $n \in \mathbb{N}$, the following square commutes:

$$\begin{tikzcd} F^{n+1}(\emptyset) \arrow[r, hook, "F(k_n)"] \arrow[d, equal] & F(I_F) \arrow[d, hook, "i_F"] \\ F^{n+1}(\emptyset) \arrow[r, hook, "k_{n+1}"'] & I_F \end{tikzcd} \tag{7}$$

The only other fact we will need to use about colimits of ω-chains in $\mathcal{D}^e$ is that they can be constructed by forming the limit in the category of posets and monotone functions of the corresponding ω^{op}-chain of projections (see [20, Theorem 2]). Hence in particular, with notation as in (4), we have

Lemma 3.2 *The family of monotone functions $(k_n^* : (I_F)_\perp \to (F^n(\emptyset))_\perp \mid n \in \mathbb{N})$ is jointly order-reflecting: that is, $u \sqsubseteq u'$ holds in $(I_F)_\perp$ if for all $n \in \mathbb{N}$, $k_n^*(u) \sqsubseteq k_n^*(u')$ holds in $(F^n(\emptyset))_\perp$.*

As well as determining ω-continuous functors $\mathcal{D}^e \to \mathcal{D}^e$, we saw in section 2 that predomain constructors come equipped with an action on binary relations. So let us now suppose that we are given an ω-continuous functor $F : \mathcal{D}^e \to \mathcal{D}^e$ equipped with a function $\lhd \in \mathcal{R}e\ell(D_\perp) \mapsto F(\lhd) \in \mathcal{R}e\ell((FD)_\perp)$ for each $D \in \mathcal{D}$. Writing $k : (I_F)_\perp \cong (F(I_F))_\perp$ for the lift of the inverse of the initial algebra structure morphism $i_F : F(I_F) \hookrightarrow I_F$ (which as we noted above is necessarily an isomorphism), let us call a relation $\lhd \in \mathcal{R}e\ell((I_F)_\perp)$ an *F-simulation* if $u \lhd u'$ implies $k(u)\ F(\lhd)\ k(u')$, for all $u, u' \in (I_F)_\perp$. The following proposition gives sufficient criteria for the initial algebra I_F to satisfy a co-induction property like that in Theorem 2.5.

Proposition 3.3 *With F as above, suppose that*

(i) *For each $D \in \mathcal{D}$, $F(\sqsubseteq_{D_\perp}) = \sqsubseteq_{(FD)_\perp}$.*

(ii) *Given* $i : D \hookrightarrow E$ *in* $\mathcal{D}^e$ *and* $\lhd \in \mathcal{R}e\ell(E_\perp)$*, suppose that* $u \lhd u'$ *implies* $i^*(u) \sqsubseteq i^*(u')$*, for all* $u, u' \in E_\perp$*. Then* $v\ F(\lhd)\ v'$ *implies* $F(i)^*(v) \sqsubseteq F(i)^*(v')$*, for all* $v, v' \in F(E)_\perp$*. (The notation* i^* *is defined in* (4)*.)*

Then for all $u, u' \in (I_F)_\perp$*,* $u \sqsubseteq u'$ *if and only if there is some* F*-simulation* $\lhd$ *with* $u \lhd u'$*.*

Proof Since (i) implies that $\sqsubseteq_{(I_F)_\perp}$ is an F-simulation, the 'only if' direction is immediate.

For the converse, suppose that $\lhd$ is an F-simulation. We have to show for all $u, u' \in (I_F)_\perp$ that $u \lhd u'$ implies $u \sqsubseteq u'$. By Lemma 3.2 it suffices to prove for all $n \in \mathbb{N}$ that

$$\text{for all } u, u' \in (I_F)_\perp, \text{ if } u \lhd u' \text{ then } k_n^*(u) \sqsubseteq k_n^*(u') \tag{8}$$

and we do this by induction on n. The base case $n = 0$ is trivial, since k_0^* has codomain $(F^0(\emptyset))_\perp = \{\perp\}$ and so is a constant function.

So suppose, inductively, that (8) holds. If $u \lhd u'$ in $(I_F)_\perp$ then $k(u)\ F(\lhd)\ k(u')$, since $\lhd$ is an F-simulation. Then applying property (ii) with $i = k_n$, we have that $F(k_n)^*(k(u)) \sqsubseteq F(k_n)^*(k(u'))$. But this is exactly $k_{n+1}^*(u) \sqsubseteq k_{n+1}^*(u')$, since k is by definition $(i_F^{-1})_\perp$ and hence from (7) we have that $F(k_n)^* \circ k = k_{n+1}^*$. Thus (8) implies the same statement for $n + 1$, as required. $\square$

Lemma 3.4 *Any functor* $F = \sigma(\_)$ *arising from a predomain constructor* σ *satisfies the hypotheses of Proposition 3.3.*

Proof Condition (i) of the proposition holds by Lemma 2.4. For condition (ii), suppose that $\lhd \in \mathcal{R}e\ell(E_\perp)$ and that $i : D \hookrightarrow E$ is an embedding such that $u \lhd u'$ implies $i^*(u) \sqsubseteq i^*(u')$, for all $u, u' \in E_\perp$. We have to prove that

$$\text{for all } v, v' \in \sigma(E)_\perp, \text{ if } v\ \sigma(\lhd)\ v' \text{ then } \sigma(i)^*(v) \sqsubseteq \sigma(i)^*(v') \tag{9}$$

and we do so by induction on the structure of σ. We give the cases for lifting and partial exponential, and omit the others. Note that in proving $\sigma(i)^*(v) \sqsubseteq \sigma(i)^*(v')$ from $v\ \sigma(\lhd)\ v'$ it suffices to consider the case $v \neq \bot$, since $\sigma(i)^*(\bot) = \bot$.

Case σ is $(\sigma_1)_\bot$: Suppose that $v\ \sigma(\lhd)\ v'$ in $\sigma(E)_\bot$, and that $v \neq \bot$—say $v = [u]$. Then since $\sigma(\lhd)$ is by definition the lifted relation $(\sigma_1(\lhd))_\bot$, $v = [u']$ for some $u' \in \sigma(E)$ satisfying $u\ \sigma_1(\lhd)\ u'$. By induction hypothesis, σ_1 satisfies (9), so $\sigma_1(i)^*(u) \sqsubseteq \sigma_1(i)^*(u')$. From the definition of $\sigma(i)^\circ$ in terms of $\sigma_1(i)^\circ$ given in Definition 3.1, it follows that $\sigma(i)^\circ(u) \sqsubseteq \sigma(i)^\circ(u')$. But since $v = [u]$, $\sigma(i)^*(v) = \sigma(i)^\circ(u)$; similarly for v' and u'. Thus $\sigma(i)^*(v) \sqsubseteq \sigma(i)^*(v')$, as required.

Case σ is $\sigma_1 \rightharpoonup \sigma_2$: Suppose that $v\ \sigma(\lhd)\ v'$ in $\sigma(E)_\bot$, and that $v \neq \bot$—say $v = [g]$ with $g \in \sigma_1(E) \rightharpoonup \sigma_2(E)$. Then by definition of $\sigma(\lhd)$, $v = [g']$ for some g' satisfying $g(y_2)\ \sigma_2(\lhd)\ g'(y_2)$ for all $y_2 \in \sigma_2(E)$. By induction hypothesis σ_2 satisfies (9), so $\sigma_2(i)^*(g(y_2)) \sqsubseteq \sigma_2(i)^*(g'(y_2))$, for all $y_2 \in \sigma_2(E)$. So for any $x_1 \in \sigma_1(D)$, taking $y_2 = \sigma_1(i)(x_1)$ we have

$$\sigma_2(i)^*(g(\sigma_1(i)(x_1))) \sqsubseteq \sigma_2(i)^*(g'(\sigma_1(i)(x_1)))$$

So from the definition of $\sigma(i)^\circ$ in terms of $\sigma_1(i)^\circ$ and $\sigma_2(i)^\circ$ given in Definition 3.1, it follows that $\sigma(i)^\circ(g) \sqsubseteq \sigma(i)^\circ(g')$. Hence $\sigma(i)^*(v) = \sigma(i)^\circ(g) \sqsubseteq \sigma(i)^\circ(g') = \sigma(i)^*(v')$, as required.

$\square$

With the above lemma, we obtain Theorem 2.5 as a direct corollary of Proposition 3.3. Indeed, we can derive the following generalization of the theorem which characterizes the partial order on a predomain built from a recursive one using the predomain constructors.

Corollary 3.5 *Suppose that σ and τ are predomain constructors as in Definition 2.1. Then $v \sqsubseteq v'$ in $\tau(\mathrm{rec}\alpha.\sigma)_\bot$ if and only if there is some σ-simulation $\lhd$ with $v\ \tau(\lhd)\ v'$.*

Proof If $v \sqsubseteq v'$, we can take $\lhd$ to be $\sqsubseteq_{(\mathrm{rec}\alpha.\sigma)_\perp}$. We know that it is a σ-simulation, and $\tau(\lhd)$ is the partial order on $\tau(\mathrm{rec}\alpha.\sigma)_\perp$ because by Lemma 3.4 property (i) in Proposition 3.3 holds for $\tau(\_)$.

For the converse, we apply property (ii) in Proposition 3.3 to $\tau(\_)$, taking $i : D \hookrightarrow E$ to be $id : \mathrm{rec}\alpha.\sigma \hookrightarrow \mathrm{rec}\alpha.\sigma$. The hypothesis in (ii) holds in this case by Theorem 2.5, and the conclusion is the result required. $\square$

4 Simultaneous Domain Equations

Let $\sigma_1, \ldots, \sigma_n$ be predomain constructors built up using $\times, +, {}_\perp, \rightharpoonup$, constants ranging over $\mathcal{D}$, and variables drawn from the set $\{\alpha_1, \ldots, \alpha_n\}$. Given an n-tuple of predomains $D = (D_1, \ldots, D_n) \in \mathcal{D}^n$, let $\sigma_i(D) \in \mathcal{D}$ denote the result of interpreting each α_j $(j = 1, \ldots, n)$ as D_j in σ_i; and let $\sigma(D) \in \mathcal{D}^n$ denote the n-tuple $(\sigma_1(D), \ldots, \sigma_n(D))$. Then D constitutes a solution to the simultaneous equations

$$\left\{ \begin{array}{ccc} \alpha_1 & = & \sigma_1 \\ & \vdots & \\ \alpha_n & = & \sigma_n \end{array} \right. \tag{10}$$

if there are isomorphism $D_i \cong \sigma_i(D)$ for each $i = 1, \ldots, n$. As before, to state the co-induction property we need only consider the lift of such isomorphisms : so call (D, k) an *invariant* for the n-tuple of predomain constructors $\sigma = (\sigma_1, \ldots, \sigma_n)$ if $k = (k_1, \ldots, k_n)$ with $k_i : (D_i)_\perp \cong \sigma_i(D)_\perp$ for each $i = 1, \ldots, n$.

A *σ-simulation* for such an invariant is an n-tuple of binary relations $\lhd = (\lhd_1, \ldots, \lhd_n)$ with $\lhd_i \in \mathcal{R}e\ell((D_i)_\perp)$ and such that for all $u, u' \in (D_1)_\perp \times \cdots \times (D_n)_\perp$, if $\forall i (u_i \lhd_i u'_i)$ then $\forall i (k_i(u_i) \; \sigma_i(\lhd) \; k_i(u'_i))$. Here we are using the natural extension of Definition 2.2 to predomain constructors ψ involving several variables $\alpha_1, \ldots, \alpha_n$: in defining $\psi(\lhd)$ by induction on the structure of σ_i, one proceeds just as in Definition 2.2, but using the appropriate component $\lhd_j$ of $\lhd$ in case ψ is a variable α_j.

Let $(\mathcal{D}^e)^n$ denote the n-fold product of the category of predomains and embeddings. $(\mathcal{D}^e)^n$ inherits a terminal object and colimits of ω-chains componentwise from $\mathcal{D}^e$. The action of a predomain constructor ψ of n variables on a morphism $i : D \to E$ in $(\mathcal{D}^e)^n$ to produce an embedding $\psi(i) : \psi(D) \hookrightarrow \psi(E)$ can be defined by induction on the structure of ψ just as in Definition 3.1. In this way the assignment $D \mapsto \psi(D)$ extends to an ω-continuous functor $\psi(\_) : (\mathcal{D}^e)^n \to \mathcal{D}^e$. And given an n-tuple $\sigma_1, \ldots, \sigma_n$ of such predomain constructors, we get an ω-continuous functor $\langle \sigma_1(\_), \ldots, \sigma_n(\_) \rangle : (\mathcal{D}^e)^n \to (\mathcal{D}^e)^n$. By definition, *the n-tuple of predomains recursively defined by (10)* is the initial algebra for this functor.

Theorem 4.1 *Let $(D_1, \ldots, D_n)$ be the n-tuple of predomains recursively defined by* (10). *Let $\tilde{D} = (D_1)_\perp \times \cdots \times (D_n)_\perp$. For any $u, u' \in \tilde{D}$, $u \sqsubseteq_{\tilde{D}} u'$ holds if and only if there is some σ-simulation $\lhd$ with $u_i \lhd_i u'_i$ for each $i = 1, \ldots, n$.*

Proof Let F_i be the functor $\sigma_i(\_) : (\mathcal{D}^e)^n \to \mathcal{D}^e$ and let $F = \langle F_1, \ldots, F_n \rangle : (\mathcal{D}^e)^n \to (\mathcal{D}^e)^n$. Then each D_i can be calculated as the colimit in $\mathcal{D}^e$ of the ω-chain

$$F_i^{(0)}(\vec{\emptyset}) \hookrightarrow F_i^{(1)}(\vec{\emptyset}) \hookrightarrow F_i^{(2)}(\vec{\emptyset}) \hookrightarrow \cdots$$

where

$$\begin{aligned} F_i^{(0)}(\vec{\emptyset}) &= \emptyset \\ F_i^{(m+1)}(\vec{\emptyset}) &= F_i(F_1^{(m)}(\vec{\emptyset}), \ldots, F_n^{(m)}(\vec{\emptyset})) \end{aligned}$$

In particular, to prove $u \sqsubseteq_{\tilde{D}} u'$ it suffices to prove that for each i and m the partial order relation holds between the projections of u_i and u'_i to $F_i^{(m+1)}(\vec{\emptyset})_\perp$. Thus the technique used in the proof of Proposition 3.3 can be applied to deduce the theorem from the following properties of the action of a predomain constructor ψ of n variables on n-tuples of binary relations:

(i) For each $D = (D_1, \ldots, D_n) \in \mathcal{D}^n$, $\psi(\sqsubseteq_{(D_1)_\perp}, \ldots, \sqsubseteq_{(D_n)_\perp}) = \sqsubseteq_{\psi(D)_\perp}$.

(ii) Given $i : D \to E$ in $(\mathcal{D}^e)^n$ and $\lhd = (\lhd_1, \ldots, \lhd_n)$ with $\lhd_j \in \mathcal{R}e\ell((E_j)_\bot)$ $(j = 1, \ldots, n)$, suppose that for all $u, u' \in (E_1)_\bot \times \cdots \times (E_n)_\bot$, if $\forall j(u_j \lhd_j u'_j)$ then $\forall j(i^*_j(u_j) \sqsubseteq_{(D_j)_\bot} i^*_j(u'_j))$. Then for all $v, v' \in \psi(D)_\bot$, if $v\ \psi(\lhd)\ v'$ then $\psi(i)^*(v) \sqsubseteq \psi(i)^*(v')$.

Properties (i) and (ii) can be proved by induction on the structure of ψ, just as in Lemma 3.4. $\square$

Remark 4.2 (Parameterized recursion) What if in Theorem 4.1 we were only interested in proving instances of the partial order in one of the D_i? To avoid constructing simulation relations for all the other components, we would have to express the simultaneous recursion of (10) in terms of single recursive domains $\mathrm{rec}\beta.\sigma(\beta, \alpha, \ldots)$, where $\alpha, \ldots$ are free parameters . A coinduction principle can be developed for such parameterized recursive types. The main step is to extend Definition 2.2 with a clause for the case that σ is $\mathrm{rec}\beta.\sigma_1(\beta, \alpha)$: one defines $\sigma(\lhd)$ to be the greatest fixed point of the monotone operator on $\mathcal{R}e\ell((\mathrm{rec}\beta.\sigma_1(\beta, D))_\bot)$ given by

$$\lhd' \quad \mapsto \quad \{(u, u') \mid k_D(u)\ \sigma_1(\lhd', \lhd)\ k_D(u')\}$$

where $k_\alpha : (\mathrm{rec}\beta.\sigma_1(\beta, \alpha))_\bot \cong \sigma_1(\mathrm{rec}\beta.\sigma_1(\beta, \alpha), \alpha)_\bot$ is the (lift of the) canonical isomorphism.

The proof that the co-induction principle does hold for parameterized recursive types is best accomplished by interpreting them not in $\mathcal{D}^e$ but in an associated ω-chain complete category $\mathcal{D}'$. The objects of $\mathcal{D}'$ are pairs $(D, \lhd)$ where D is a predomain and $\lhd \in \mathcal{R}e\ell(D_\bot)$. A morphism $(D, \lhd) \to (D', \lhd')$ is defined to be an embedding $i : D \hookrightarrow D'$ satisfying $v \lhd' v'$ implies $i^*(v) \lhd i^*(v')$, for all $v, v' \in D'_\bot$. Colimits of ω-chains in $\mathcal{D}'$ involve forming colimits of ω-chains of embeddings in $\mathcal{D}^e$ and taking the greatest fixed point of a certain monotone operator on relations to get the second component of the colimit. There is an ω-continuous retraction from $\mathcal{D}'$ to $\mathcal{D}^e$, and the predomain constructors $_\bot, \times, +, \rightharpoonup$ can be lifted along it to $\mathcal{D}'$.[1] These properties enable one to extract the co-induction property for parameterized recursive predomains in $\mathcal{D}^e$ from an analysis of initial algebras for ω-continuous functors on $\mathcal{D}'$. We leave the fleshing out of the details of this to another occasion. (The category $\mathcal{D}'$ should be contrasted with the one involving ω-chain complete relations mentioned in [20, Section 4, Example 6]: for co-induction (as opposed to induction) we are in the pleasant position of not needing to impose chain completeness conditions.)

5 Co-induction with Powerdomains

We saw in Example 2.7 that the notion of 'applicative bisimulation' used by Abramsky in his study of the lazy lambda calculus [1] is a particular case of the notion of simulation for a predomain constructor. The same is true for the notion of 'partial bisimulation' in Abramsky's denotational semantics of SCCS [2]. To see this we must enrich our collection of predomain constructors to include the convex, or Plotkin, powerdomain (or rather, the predomain version of it). In order to prove that the co-induction property continues to hold for predomains recursively defined using this larger set of constructors, we will need to restrict attention to the bifinite predomains, for which there is a sufficiently concrete description of the convex powerdomain construction.

We say that $D \in \mathcal{D}$ is *bifinite* if it is the colimit in $\mathcal{D}^e$ of an ω-chain of finite predomains. Let $\mathcal{B}$ denote the collection of bifinite predomains, and let $\mathcal{B}^e$ denote the full subcategory of $\mathcal{D}^e$ whose objects are in $\mathcal{B}$. In section 3 we remarked that $\mathcal{D}^e$ is isomorphic, via the lifting functor, to the category of 'embedding-projection pairs' between ω-complete posets with least elements. This isomorphism restricts to one between $\mathcal{B}^e$ and the usual category of bifinite (or 'SFP' [17]) domains and embedding-projection pairs. So given $D \in \mathcal{B}$, $D_\perp$ is a bifinite domain and we have

[1] The author does not know whether the power-predomain constructor P considered in section 5 can be so lifted.

from [18, Chapter 8] the following explicit description of the convex powerdomain of $D_\perp$, $P^\natural(D_\perp)$:

- The underlying set of $P^\natural(D_\perp)$ consists of all non-empty, *convex*, *Lawson-closed* subsets of $D_\perp$. Recall that $X \subseteq D_\perp$ is convex if whenever $x_1 \sqsubseteq x \sqsubseteq x_2$ with $x_1, x_2 \in X$, then $x \in X$. A subset X of bifinite $D_\perp$ is Lawson-closed if X contains the limit of any convergent sequence $(x_n \mid n \in \mathbb{N})$ contained in X. (By definition, the sequence $(x_n \mid n \in \mathbb{N})$ converges to a limit x if for all compact elements c, if $c \sqsubseteq x$ (respectively $c \not\sqsubseteq x$) then $c \sqsubseteq x_n$ (respectively $c \not\sqsubseteq x_n$) for all but finitely many n.)

- The partial order on $P^\natural(D_\perp)$ is the *Egli-Milner* order, $\sqsubseteq_{EM}$: given $X, X' \in P^\natural(D_\perp)$, $X \sqsubseteq_{EM} X'$ is defined to hold if and only if
  - for all $x \in X$ there is some $x' \in X'$ with $x \sqsubseteq_{D_\perp} x'$; and
  - for all $x' \in X'$ there is some $x \in X$ with $x \sqsubseteq_{D_\perp} x'$.

The convex powerdomain construction extends to a (locally continuous) functor on the category of bifinite domains and continuous functions, and hence to an ω-continuous functor on the category of bifinite domains and embedding-projection pairs. The action of $P^\natural$ on a continuous function $p : B \to A$ between bifinite domains produces the continuous function $P^\natural(p) : P^\natural(B) \to P^\natural(A)$ mapping a non-empty, convex, Lawson-closed subset $Y \subseteq B$ to the convex hull of the Lawson closure of the image of Y along p (see [2, Section 3]). We record the following facts that we need below and which can be obtained from [18, Chapter 8].

Lemma 5.1

(i) *Let $Con(X)$ denote the* convex hull *of a subset X of a predomain D: thus by definition $Con(X) = \{x \in D \mid x_1 \sqsubseteq x \sqsubseteq x_2, \text{for some } x_1, x_2 \in X\}$. If $Y \subseteq D$ is convex, then $Con(X) \sqsubseteq_{EM} Y$ if and only if $X \sqsubseteq_{EM} Y$.*

(ii) *For a bifinite domain B, the least element of $P^\natural(B)$ is $\{\bot\}$.*

(iii) *Let $p : B \to A$ be the projection half of an embedding-projection pair between bifinite domains. (In other words, p possesses a right-inverse-left-adjoint.) Then the image of a Lawson-closed subset of B along p is again Lawson-closed. Hence $P^\natural(p)$ sends $Y \in P^\natural(B)$ to*

$$P^\natural(p)(Y) \quad = \quad Con(\{p(y) \mid y \in Y\})$$

Turning now to a predomain version of the convex powerdomain, we can restrict $P^\natural$ along the isomorphism (given by lifting) of $\mathcal{B}^e$ with the category of bifinite domains and embedding-projection pairs, to obtain an ω-continuous functor $P : \mathcal{B}^e \to \mathcal{B}^e$. Thus the object part of this functor sends $D \in \mathcal{B}$ to the bifinite predomain $P(D)$ with the property that $P(D)_\bot = P^\natural(D_\bot)$. Given an embedding $i : D \hookrightarrow E$ in $\mathcal{B}^e$, then using Lemma 5.1(iii), the embedding $P(i) : P(D) \hookrightarrow P(E)$ has an associated partial projection $P(i)^\circ$ such that for all $V \in P(E)_\bot$

$$P(i)^*(V) \quad = \quad Con(\{i^*(v) \mid v \in V\})$$

(where $P(i)^*$ is defined from $P(i)^\circ$ as in (4)).

We also have to give an action of this power-predomain constructor on binary relations. Given $D \in \mathcal{B}$ and $\lhd \in \mathcal{R}e\ell(D_\bot)$, define $P(\lhd) \in \mathcal{R}e\ell(P(D)_\bot)$ as follows: for $U, U' \in P(D)_\bot = P^\natural(D_\bot)$, we define $U \; P(\lhd) \; U'$ to hold if and only if:

- for all $u \in U$ there is some $u' \in U'$ with $u \lhd u'$; and
- for all $u \in U'$ there is some $u \in U$ with $u \lhd u'$.

Lemma 5.2

(i) *For each $D \in \mathcal{B}$, $P(\sqsubseteq_{D_\bot}) = \sqsubseteq_{P(D_\bot)}$.*

(ii) *Given* $i : D \hookrightarrow E$ *in* $\mathcal{B}^e$ *and* $\lhd \in \mathcal{R}e\ell(E_\perp)$, *suppose that* $v \lhd v'$ *implies* $i^*(v) \sqsubseteq i^*(v')$, *for all* $v, v' \in E_\perp$. *Then for all* $V, V' \in P(E)_\perp$, *if* $V \; P(\lhd) \; V'$ *then* $P(i)^*(V) \sqsubseteq P(i)^*(V')$.

Proof For (i), just note that in the definition of $P(\lhd)$ if $\lhd$ is the partial order $\sqsubseteq_{D_\perp}$ then $P(\lhd)$ is the Egli-Milner ordering, which we noted above is the partial order for $P(D)_\perp = P^\natural(D_\perp)$ when $D \in \mathcal{B}$.

For (ii), suppose that $V \; P(\lhd) \; V'$. We have to show that $P(i)^*(V) \sqsubseteq P(i)^*(V')$, i.e. that $Con(\{i^*(v) \mid v \in V\}) \sqsubseteq_{EM} Con(\{i^*(v') \mid v' \in V'\})$. By Lemma 5.1(i), it is sufficient to show that $\{i^*(v) \mid v \in V\} \sqsubseteq_{EM} Con(\{i^*(v') \mid v' \in V'\})$. By definition of the Egli-Milner order, this requires us to prove

(a) for all $v \in V$, there is $u \in Con(\{i^*(v') \mid v' \in V'\})$ with $i^*(v) \sqsubseteq u$; and

(b) for all $u \in D_\perp$ and $v'_1, v'_2 \in V'$, if $i^*(v'_1) \sqsubseteq u \sqsubseteq i^*(v'_2)$, then there is $v \in V$ with $i^*(v) \sqsubseteq u$.

For (a), recalling the definition of $P(\lhd)$, if $v \in V$ and $V \; P(\lhd) \; V'$ then there is some $v' \in V'$ with $v \lhd v'$ and hence, by hypothesis on i, $i^*(v) \sqsubseteq i^*(v')$. Since $i^*(v') \in Con(\{i^*(v') \mid v' \in V'\})$ we are done.

For (b), suppose $i^*(v'_1) \sqsubseteq u \sqsubseteq i^*(v'_2)$ with $v'_1, v'_2 \in V'$. Then since $V \; P(\lhd) \; V'$ and $v'_1 \in V'$, there is some $v \in V$ with $v \lhd v'_1$, and hence $i^*(v) \sqsubseteq i^*(v'_1) \sqsubseteq u$ as required. $\square$

Let us extend the notion of predomain constructor given in Definition 2.1 by permitting P to be used:

$$\sigma \quad ::= \quad \cdots \mid P(\sigma)$$

Interpreting the constants K that occur in such a σ as ranging over $\mathcal{B}$, each such predomain constructor determines an ω-continuous functor $\sigma(\_) : \mathcal{B}^e \to \mathcal{B}^e$ and an initial algebra $\mathrm{rec}\alpha.\sigma \in \mathcal{B}$. Using the definition of $\lhd \mapsto P(\lhd)$ given above, we can extend Definition 2.2 to give an action of such a predomain constructor σ on binary relations on predomains in $\mathcal{B}$. In particular we have the notion of a σ-simulation on $\mathrm{rec}\alpha.\sigma$, just as in Definition 2.3. By Lemma 5.2, the action on binary relations satisfies the hypotheses of Proposition 3.3. So that proposition yields the following extension of Theorem 2.5.

Theorem 5.3 (Co-induction with Powerdomains) *Let σ be a predomain constructor of a single variable α built up using $\times, +, {}_\perp, \rightharpoonup, P$ and constants ranging over bifinite predomains. Let $D = \mathrm{rec}\alpha.\sigma$ be the bifinite predomain recursively defined by the equation $\alpha = \sigma$. For any $u, u' \in D_\perp$, $u \sqsubseteq_{D_\perp} u'$ holds if and only if there is some σ-simulation $\lhd$ with $u \lhd u'$.*

Example 5.4 Abramsky's domain for bisimulation in [2] can be expressed with predomain constructors as $B = (\mathrm{rec}\alpha.\sigma)_\perp$ with $\sigma = 1 + P(\mathbb{N} \times \alpha_\perp)$. Let $k : B \cong (1 + P(\mathbb{N} \times B))_\perp$ be the associated lifted isomorphism. Then from the definitions we have that a binary relation $\lhd$ on B is a σ-simulation if and only if

- if $u \lhd u'$ and $k(u) = [\mathrm{inl}(0)]$, then $k(u') = [\mathrm{inl}(0)]$; and
- if $u \lhd u'$ and $k(u) = [\mathrm{inr}(x)]$ for some $x \in P(\mathbb{N} \times B)$ (so that $[x] \in P^\natural((\mathbb{N} \times B)_\perp)$), then $k(u') = [\mathrm{inr}(x')]$ for some $x' \in P(\mathbb{N} \times B)$ satisfying
  - for all $v \in [x]$ there is $v' \in [x']$ with $v \lhd' v'$, and
  - for all $v' \in [x']$ there is $v \in [x]$ with $v \lhd' v'$

  where $\lhd' \in \mathcal{R}e\ell((\mathbb{N} \times B)_\perp)$ is constructed from $\lhd$ by defining $v \lhd' v'$ to hold if and only if

  > for all $n \in \mathbb{N}$ and $u \in B$, if $v = [(n, u)]$ then $v' = [(n, u')]$ for some $u' \in B$ with $u \lhd u'$.

The notion of σ-simulation in this case can be simplified. Following [2], let us introduce the following notation, where $u, u' \in B$ and $n \in \mathbb{N}$:

- Write $u \xrightarrow{n} u'$ to mean that $k(u) = [\mathrm{inr}(x)]$ for some $x \in P(\mathbb{N} \times B)$ with $[(n, u')] \in [x]$.
- Write $u{\uparrow}$ to mean that either $k(u) = \bot$, or $k(u) = [\mathrm{inr}(x)]$ for some $x \in P(\mathbb{N} \times B)$ with $\bot \in [x]$.
- Write $u{\downarrow}$ to mean that $u{\uparrow}$ does not hold.

Then it is straightforward (if somewhat tedious) to prove:

Proposition 5.5 *With σ and B as above, a binary relation $\lhd$ on B is a σ-simulation if and only if it is a* partial bisimulation *in the sense of Abramsky [2], i.e. if and only if it satisfies:*

(i) *for all $u, u', v \in B$ and $n \in \mathbb{N}$, if $u \lhd v$ and $u \xrightarrow{n} u'$, then $v \xrightarrow{n} v'$ for some $v' \in B$ with $u' \lhd v'$; and*

(ii) *for all $u, v \in B$, if $u \lhd v$ and $u{\downarrow}$ then*

  $v{\downarrow}$, and

  for all $v' \in B$ and $n \in \mathbb{N}$, if $v \xrightarrow{n} v'$ then $u \xrightarrow{n} u'$ for some u' with $u' \lhd v'$.

Since partial bisimulations are exactly $(1 + P(\mathbb{N} \times \alpha_\bot))$-simulations, Abramsky's 'internal full abstraction' result [2, Proposition 3.11] for B is a special case of the co-induction property established in Theorem 5.3.

6 ML Polymorphism

In this section we show how the co-induction property can be extended to recursively defined domain *constructors*. We will revert to using just $\times, +, {}_\bot, \rightharpoonup$, although the convex powerdomain could be included without difficulty.

A well known feature of Standard ML is the ability it gives the programmer to declare datatypes parameterized by type variables. The datatype of polymorphic lists

$$\texttt{datatype}\quad \alpha\texttt{list} \;\; = \;\; \texttt{Nil | Cons of } \alpha \texttt{ * } \alpha\texttt{list}$$

is the prototypical example in practice. However, the presence of ML-style polymorphism permits not only the declaration of such type-indexed families of recursively defined types, but also recursively defined functions from types to types, such as

$$\texttt{datatype}\quad \alpha\texttt{ty} \;\; = \;\; \texttt{Nil | Cons of (}\alpha \texttt{ * } \alpha\texttt{ty)ty} \tag{11}$$

If one is using predomains to denote types, then the denotation of `ty` should be a function $F : \mathcal{D} \to \mathcal{D}$ satisfying

$$F \;\; = \;\; \lambda D \in \mathcal{D} \,.\, 1 + F(D \times F(D)) \tag{12}$$

The mathematical framework of [20] provides sufficient tools for establishing the existence of (initial) solutions to functional equations such as (12). Recall from section 3 that $\mathcal{D}^e$ denotes the category of predomains and embeddings. Let $\mathcal{C}ts(\mathcal{D}^e)$ denote the category whose objects are ω-continuous functors $\mathcal{D}^e \to \mathcal{D}^e$ and whose morphisms are natural transformations between such functors. $\mathcal{C}ts(\mathcal{D}^e)$ has colimits of ω-chains, created pointwise from $\mathcal{D}^e$: the colimit of

$$\Phi_0 \to \Phi_1 \to \cdots \tag{13}$$

in $\mathcal{C}ts(\mathcal{D}^e)$ is calculated by evaluating (13) at each $D \in \mathcal{D}$

$$\Phi_0(D) \hookrightarrow \Phi_1(D) \hookrightarrow \cdots \tag{14}$$

taking the colimit ΦD of this chain of embeddings, and using the universal property of these colimits to extend $D \mapsto \Phi D$ to a functor, which is necessarily ω-continuous. Note in particular that the *evaluation functor*

$$\begin{array}{ccc} \mathcal{C}ts(\mathcal{D}^e) \times \mathcal{D}^e & \to & \mathcal{D}^e \\ (F, D) & \mapsto & F(D) \end{array} \tag{15}$$

is jointly ω-continuous.

As well as possessing colimits of ω-chains, $\mathcal{C}ts(D^e)$ has an initial object, namely the constant functor with value $\emptyset$. It follows that an ω-continuous functors $\Phi : \mathcal{C}ts(\mathcal{D}^e) \to \mathcal{C}ts(\mathcal{D}^e)$ possesses initial algebra I_Φ. The right-hand side of (12) is the object part of a functor Φ, and by definition the predomain constructor recursively defined by (12) is (the underlying object part of) the initial algebra I_Φ in $\mathcal{C}ts(D^e)$.

Turning from this specific example to the general case, let us extend the syntax of predomain constructors given in Definition 2.1:

$$\sigma \quad ::= \quad \cdots \mid \kappa(\sigma) \tag{16}$$

where κ is a (*unary*) *constructor symbol*. For simplicity we will consider such predomain constructors involving at most one such constructor symbol κ and at most one variable α. Given a function $F : \mathcal{D} \to \mathcal{D}$ mapping predomains to predomains, and a predomain $D \in \mathcal{D}$, let $\sigma(F, D) \in \mathcal{D}$ denote the result of replacing κ by F and α by D in σ (and interpreting $_\perp$, $\times$, $+$, $\rightharpoonup$ and constants as usual).

In order to develop a notion of simulation for this kind of predomain constructor, we must give a suitable version of Definition 2.2.

Definition 6.1 Given $F : \mathcal{D} \to \mathcal{D}$ and a $\mathcal{D}$-indexed family of binary relations $\lhd = (\lhd_D \in \mathcal{R}e\ell(F(D)_\perp) \mid D \in \mathcal{D})$, for each σ as in (16) define a family of binary relations

$$\sigma(\lhd) = (\sigma(\lhd)_D \in \mathcal{R}e\ell(\sigma(F, D)_\perp) \mid D \in \mathcal{D})$$

by induction on the structure of σ as follows:

Case σ is $\kappa(\sigma_1)$: $\sigma(\lhd)_D$ is $\lhd_{\sigma_1(F,D)}$.

Case σ is α: $\sigma(\lhd)_D$ is $\sqsubseteq_{D_\perp}$. (Note that this is different from the corresponding clause in Definition 2.2: in section 2 the variable α stood for a predomain to be given a recursive definition; here that role is played by $\kappa(\alpha)$, whereas α itself just acts as a parameter.)

Case σ is K, $(\sigma_1)_\perp$, or $\sigma_1 * \sigma_2$ $(* = \times, +, \rightharpoonup)$: these cases are dealt with exactly as in Definition 2.2.

By an *invariant* for σ we mean a pair (F, k) where $F : \mathcal{D} \to \mathcal{D}$ and k is a family of isomorphisms, $k = (k_D : F(D)_\perp \cong \sigma(F, D)_\perp \mid D \in \mathcal{D})$.

Definition 6.2 A *σ-simulation* for the invariant (F, k) is a family of binary relations $\lhd = (\lhd_D \in \mathcal{R}e\ell(F(D)_\perp) \mid D \in \mathcal{D})$ such that for all $D \in \mathcal{D}$ and all $u, u' \in F(D)_\perp$, if $u \lhd_D u'$ then $k^*_D(u)\ \sigma(\lhd)_D\ k^*_D(u')$.

Example 6.3 The predomain constructor corresponding to the ML datatype declaration (11) is $\sigma = 1 + \kappa(\alpha \times \kappa(\alpha))$. Suppose (F, k) is an invariant for this σ. For $D \in \mathcal{D}$ and $u \in F(D)_\perp$, write $u \Downarrow \mathit{nil}$ if $k_D(u) = [\mathrm{inl}(0)]$, and write $u \Downarrow x$ if $k_D(u) = [\mathrm{inr}(x)]$ for some $x \in F(D \times F(D))$. Then a σ-simulation for (F, k) is a family of binary relations $(\lhd_D \in \mathcal{R}e\ell(F(D)_\perp) \mid D \in \mathcal{D})$ satisfying:

- if $u \lhd_D u'$ and $u \Downarrow \mathit{nil}$, then $u' \Downarrow \mathit{nil}$; and
- if $u \lhd_D u'$ and $u \Downarrow x$, then $u' \Downarrow x'$ for some x' with $[x] \lhd_{F(D \times F(D))} [x']$.

Given σ as in (16), say that an invariant (F, k) is *extensional* if it has the following co-induction property: for all $D \in \mathcal{D}$ and all $u, u' \in F(D)_\perp$, $u \sqsubseteq u'$ holds if and only if there is some σ-simulation $\lhd$ with $u \lhd_D u'$. We aim to show that the predomain constructor recursively defined by the equation $\kappa(\alpha) = \sigma$ is extensional. To do so we must extend the material on embeddings in section 3.

Definition 6.4 Let $i : F \to G$ be a natural transformation between ω-continuous functors $\mathcal{D}^e \to \mathcal{D}^e$, and let $j : D \hookrightarrow E$ be an embedding between predomains. For each σ as in (16) define an embedding

$$\sigma(i, j) : \sigma(F, D) \hookrightarrow \sigma(G, E)$$

by induction on the structure of σ, as follows:

Case σ is $\kappa(\sigma_1)$: $\sigma(i,j)$ is the composition

$$F(\sigma_1(F,D)) \overset{i_{\sigma_1(F,D)}}{\hookrightarrow} G(\sigma_1(F,D)) \overset{G(\sigma_1(i,j))}{\hookrightarrow} G(\sigma_1(G,E))$$

(By naturality of i, this is equal to the composition $i_{\sigma_1(G,E)} \circ F(\sigma_1(i,j))$.)

Case σ is K, α, $(\sigma_1)_\perp$, or $\sigma_1 * \sigma_2$ ($* = \times, +, \rightharpoonup$): these cases are dealt with exactly as in Definition 3.1.

Lemma 6.5 *Suppose that σ is a predomain constructor as in* (16)*, $G : \mathcal{D} \to \mathcal{D}$ is an ω-continuous functor, and $\lhd = (\lhd_D \in \mathcal{R}e\ell(G(D)_\perp) \mid D \in \mathcal{D})$ is a $\mathcal{D}$-indexed family of binary relations.*

(i) *If $\lhd = (\sqsubseteq_{G(D)_\perp} \mid D \in \mathcal{D})$ then for each $D \in \mathcal{D}$, $\sigma(\lhd)_D$ is the partial order on $\sigma(G,D)_\perp$.*

(ii) *Suppose $i : F \to G$ is a morphism in $\mathcal{C}ts(\mathcal{D}^e)$ such that for all $D \in \mathcal{D}$ and all $u, u' \in G(D)_\perp$, if $u \lhd_D u'$ then $i^*_D(u) \sqsubseteq i^*_D(u')$ in $F(D)_\perp$. Then for all $D \in \mathcal{D}$ and all $v, v' \in \sigma(G,D)_\perp$, if $v\ \sigma(\lhd)_D\ v'$ then $\sigma(i, id_D)^*(v) \sqsubseteq \sigma(i, id_D)^*(v')$ in $\sigma(F,D)_\perp$.*

Proof These properties can be proved by induction on the structure of σ using Definitions 6.1 and 6.4 together with the corresponding properties of $_\perp$, $\times$, $+$ and $\rightharpoonup$ established for the monomorphic case in Lemma 3.4. $\square$

We noted above that the evaluation functor (15) is ω-continuous. From this and the ω-continuity of the constructors $_\perp, \times, +, \rightharpoonup$ it follows that the assignment $(F,D) \mapsto \sigma(F,D)$ is the object part of an ω-continuous functor $\mathcal{C}ts(\mathcal{D}^e) \times \mathcal{D}^e \to \mathcal{D}^e$ whose action on morphisms is given by Definition 6.4. Currying this functor, we obtain an ω-continuous functor $\Phi : \mathcal{C}ts(\mathcal{D}^e) \to \mathcal{C}ts(\mathcal{D}^e)$. By definition, *the predomain constructor recursively defined by the equation $\kappa(\alpha) = \sigma$* is the (underlying object part of the) initial algebra I in $\mathcal{C}ts(\mathcal{D}^e)$ for this functor. Thus $I : \mathcal{D}^e \to \mathcal{D}^e$ is an ω-continuous functor that comes equipped with a natural transformation

$\Phi(I) \to I$ making (I, i) initial amongst such data. As usual, initiality entails that i is actually a natural isomorphism. Putting $k_D = (i_D^{-1})_\perp$, we get an invariant (I, k) for σ which is indeed extensional in the sense defined above.

Theorem 6.6 (Co-induction for recursively defined constructors) *Let σ be a predomain constructor possibly involving $_\perp$, $\times$, $+$, $\rightharpoonup$, constants from $\mathcal{D}$, a single (unary) constructor symbol κ, and a single variable α. Let $I : \mathcal{D} \to \mathcal{D}$ be the predomain constructor recursively defined by the equation $\kappa(\alpha) = \sigma$. Then for any $D \in \mathcal{D}$ and any $u, u \in I(D)_\perp$, $u \sqsubseteq_{I(D)_\perp} u'$ holds if and only if there is some σ-simulation $\lhd$ with $u \lhd_D u'$.*

Proof The 'only if' direction follows from Lemma 6.5(i) since that result implies that the family $(\sqsubseteq_{I(D)_\perp} \mid D \in \mathcal{D})$ is a σ-simulation.

For the converse, first note that since I is the inital algebra for the functor $\Phi : \mathcal{C}ts(\mathcal{D}^e) \to \mathcal{C}ts(\mathcal{D}^e)$ described above, it can be constructed as the colimit in $\mathcal{C}ts(\mathcal{D}^e)$ of the ω-chain

$$\Phi^{(0)} \xrightarrow{i_0} \Phi^{(1)} \xrightarrow{i_1} \cdots$$

where

$$\begin{array}{rcl} \Phi^{(0)}(\_) & = & \emptyset \\ \Phi^{(n+1)}(\_) & = & \sigma(\Phi^{(n)}, \_) \end{array}$$

Let $(k_n : \Phi^{(n)} \to I \mid n \in \mathbb{N})$ be the colimit cone for this chain. Since such colimits in $\mathcal{C}ts(\mathcal{D}^e)$ can be calculated pointwise from $\mathcal{D}$, for each $D \in \mathcal{D}$ $(k_{n,D} : \Phi^{(n)}(D) \hookrightarrow I(D) \mid n \in \mathbb{N})$ is a colimit cone in $\mathcal{D}^e$ for the chain

$$\Phi^{(0)}(D) \overset{i_{0,D}}{\hookrightarrow} \Phi^{(1)}(D) \overset{i_{1,D}}{\hookrightarrow} \cdots$$

In particular $u \sqsubseteq u'$ holds in $I(D)_\perp$ if and only if for each n, $k_{n,D}^*(u) \sqsubseteq k_{n,D}^*(u')$ holds in $\Phi^{(n)}(D)_\perp$. So to prove the required result it suffices to show for each $n \in \mathbb{N}$ that

$$\text{for all } D \in \mathcal{D} \text{ and all } u, u' \in I(D)_\perp, \text{ if } u \lhd_D u' \text{ then } k^*_{n,D}(u) \sqsubseteq k^*_{n,D}(u')$$

Just as for the monomorphic case in the proof of Proposition 3.4, this can be proved by induction on n using Lemma 6.5(ii). □

Remark 6.7 Theorem 6.6 readily extends to give a co-induction property for simultaneously defined recursive predomain constructors (along the lines of section 4). Similarly, using the material in section 5, the property continues to hold when σ involves the convex power-predomain construction.

7 Conclusion

Recursively defined domains, in their full generality, have a deserved reputation for being difficult to analyse. The co-induction property given in this paper appears to be a useful tool for working with them. It is particularly pleasing that the property can be stated and used without recourse to any of the explicit constructions of these domains that are available in the literature. However, to confirm this promising appearance more experience with applying the principle is needed—especially in the area of lazy datatypes.

Although it may not be apparent, the results presented here arose from studying Freyd's work on *algebraically compact categories* [5]. A important aspect of that work is the emphasis it puts on the fact that recursive datatypes arising from *functorial* constructors should be modelled by objects that are simultaneously initial algebras and final coalgebras. A predomain constructor σ is functorial if for example it contains no negative occurrences of α. For such a σ our co-induction property is in fact equivalent to the uniqueness part of the final coalgebra property.[2] In particular, in this restricted case the co-induction property is sufficient to characterize $\mathrm{rec}\alpha.\sigma$ uniquely up to isomorphism.

[2]The existence part of the coalgebra property follows from the existence of (least) fixed points for continuous functions between domains.

When σ does contain negative occurrences of α the situation is not so nice. For example, consider $\sigma = \alpha \rightharpoonup \mathbb{N}$. It is not hard to see that for this σ the monotone operator $\mathcal{R}e\ell(D_\perp) \to \mathcal{R}e\ell(D_\perp)$ associated with an invariant (D, k) as in section 2, is constant with value $\sqsubseteq_{D_\perp}$. Consequently *any* invariant is extensional in the sense defined in that section. In particular, the co-induction property does not serve to characterize $\mathrm{rec}\alpha.\sigma$ amongst invariants in this case.

This weakness stems from the way Definition 2.2 treats the partial function constructor, by throwing away the relation in the negative part of $\sigma_1 \rightharpoonup \sigma_2$. To obtain a more refined action on relations, whilst still producing monotone operators, one can adapt another important idea from [5]. Type constructors containing both positive and negative occurrences of type variables are neither co- nor contra-variant functors on a sutiable category $\mathcal{C}$ of types and functions; however, they can be viewed as diagonalized versions of functors on $\mathcal{C}^{\mathrm{op}} \times \mathcal{C}$. Adapting this idea to the concerns of this paper leads to an interesing proof principle for recursive domains that contains both induction and co-induction principles simultaneously; this is described in [16].

Acknowledgements

The research described in this paper was supported by the ESPRIT 'CLICS' Basic Research Project and UK SERC grant GR/G53279. It benefited from the comments of Samson Abramsky, Mike Fourman, Wesley Phoa, Gordon Plotkin, Steve Vickers (who suggested Example 2.6), the comments of an anonymous referee, and members of the CLICS club in Cambridge.

References

[1] S. Abramsky, The Lazy Lambda Calculus, in: D. Turner, ed., *Research Topics in Functional Programming* (Addison-Wesley, Reading MA, 1990) 65–116.

[2] S. Abramsky, A Domain Equation for Bisimulation, *Information and Computation* **92**(1991) 161–218.

[3] S. Abramsky and C.-H. L. Ong, Full Abstraction in the Lazy Lambda Calculus, *Information and Computation*, to appear.

[4] R. Bird and P. Wadler, *Introduction to Functional Programming* (Prentice Hall International, 1988).

[5] P. J. Freyd, Algebraically Complete Categories, in: A. Carboni *et al*, eds, *Proceedings of the 1990 Como Category Theory Conference*, Lecture Notes in Mathematics, Vol. 1488 (Springer, Berlin, 1991) 95–104.

[6] D. Howe, Equality in Lazy Computation Systems, in [8] 198–203.

[7] P. Hudak, S. Peyton Jones and P. Wadler, eds, Report on the Programming Language Haskell: Version 1.1, Technical Report, Yale University and Glasgow University, August 1991.

[8] *Proc. 4th Annual Symposium on Logic in Computer Science, Asilomar CA* (IEEE Computer Society Press, Washington, 1989).

[9] R. Milner, *Communication and Concurrency* (Prentice-Hall, 1989).

[10] R. Milner and M. Tofte, Co-induction in relational semantics, *Theoretical Computer Science* **87**(1991) 209–220.

[11] R. Milner, M. Tofte and R. Harper, *The Definition of Standard ML* (MIT Press, 1990).

[12] E. Moggi, Computational lambda-calculus and monads, in [8] 14–23.

[13] E. Moggi, An Abstract View of Programming Languages, Lecture Notes, Stanford University, July 1989.

[14] D. Park, Fixpoint induction and proofs of program properties, *Machine Intelligence* **5**(1970) 59–78.

[15] L. C. Paulson, *ML for the Working Programmer* (Cambridge University Press, 1991).

[16] A. M. Pitts, Relational Properties of Recursively Defined Datatypes, in *Proc. 8th Annual Symposium on Logic in Computer Science, Montréal* (IEEE Computer Society Press, Washington, 1993) 86–97.

[17] G. D. Plotkin, A Powerdomain Construction, *SIAM J. Comput.* **5**(1976) 452–487.

[18] G. D. Plotkin, Lecture Notes for Postgraduate Course on Domain Theory, Dept. of Computer Science, University of Edinburgh, 1981/82.

[19] G. D. Plotkin, *Lectures on Predomains and Partial Functions.* Notes for a course at CSLI, Stanford University, 1985.

[20] M. B. Smyth and G. D. Plotkin, The Category-Theoretic Solution of Recursive Domain Equations, *SIAM J. Comput.* **11**(1982) 761–783.

[21] G. Winskel and K. G. Larsen, Using Information Systems to Solve Recursive Domain Equations Effectively, in: G. Kahn *et al*, eds, *Semantics of Data Types*, Lecture Notes in Computer Science, Vol. 173 (Springer, Berlin, 1984) 109–130.

List of Footnotes

1. The author does not know whether the power-predomain constructor P considered in section 5 can be so lifted.

2. The existence part of the coalgebra property follows from the existence of (least) fixed points for continuous functions between domains.