Math. Struct. in Comp. Science (2009), *vol.* 19, *pp.* 639–686. © 2009 Cambridge University Press
doi:10.1017/S096012950900766X First published online 6 May 2009 Printed in the United Kingdom

From parametric polymorphism to models of polymorphic FPC

RASMUS EJLERS MØGELBERG[†]

IT University of Copenhagen, Rued Langgaards Vej 7, 2300 Copenhagen S, Denmark
Email: mogel@itu.dk

Received 26 November 2006; revised 9 December 2008

This paper shows how PILL$_Y$ (Polymorphic Intuitionistic/Linear Lambda calculus with a fixed point combinator Y) with parametric polymorphism can be used as a metalanguage for domain theory, as originally suggested by Plotkin more than a decade ago. Using Plotkin's encodings of recursive types in PILL$_Y$, we show how parametric models of PILL$_Y$ give rise to models of FPC, which is a simply typed lambda calculus with recursive types and an operational call-by-value semantics, reflecting a classical result from domain theory. Essentially, this interpretation is an interpretation of intuitionistic logic into linear logic first discovered by Girard, which in this paper is extended to deal with recursive types.
Of particular interest is a model based on 'admissible' pers over a reflexive domain, the theory of which can be seen as a domain theory for (impredicative) polymorphism. We show how this model gives rise to a parametric and computationally adequate model of PolyFPC, an extension of FPC with impredicative polymorphism. This is to the author's knowledge the first denotational model of a non-linear language with parametric polymorphism and recursive types.

1. Introduction

Parametric polymorphism is an important reasoning principle for several reasons. One is that it provides proofs of modularity principles (Reynolds 1983) and other results based on 'information hiding' such as security principles (see, for example, Tse and Zdancewic (2004)). Another is that it can be used to make simple type theories surprisingly expressive by encoding inductive and coinductive types using polymorphism. If, in addition, parametric polymorphism is combined with fixed points at the term level, inductive and coinductive types coincide, and Freyd's theory of algebraically compact categories (Freyd 1990b; Freyd 1990a; Freyd 1991) provides solutions to general type equations. However, when introducing fixed points, the parametricity principle must be weakened for the theory to be consistent. Plotkin (Plotkin 1993a; Plotkin 1993b) suggested using the calculus PILL$_Y$ (Polymorphic dual Intuitionistic/Linear Lambda calculus with a fixed point combinator Y), which in combination with parametricity would have recursive types in the linear part of the calculus. This theory was worked out in detail, along with a category theoretic treatment, by Birkedal, Møgelberg and Petersen (see Birkedal *et al.*

[†] The work here was conducted while the author was associated with Università di Genova, Italy and was sponsored by Danish Research Agency stipend no. 272-05-0031.

(2006a), Birkedal *et al.* (2008) and Birkedal *et al.* (2006b); see also Møgelberg (2005)). In Birkedal *et al.* (2007), a concrete model of PILL_Y was constructed using 'admissible' partial equivalence relations (pers) over a reflexive domain. The theory of admissible pers can be viewed as a domain theory for (impredicative) polymorphism.

Plotkin suggested using PILL_Y with parametric polymorphism as an axiomatic setup for domain theory. However, as mentioned, the solutions to recursive type equations that PILL_Y provides are in a linear calculus, whereas, as is well known, domain theory also provides models of non-linear lambda calculi with recursive types, such as FPC, which is a simply typed lambda calculus with general recursive types equipped with an operational call-by-value semantics. In this paper we test Plotkin's thesis by showing that the solutions to recursive type equations in the linear type theory PILL_Y can be used to model FPC. The resulting translation of FPC into PILL_Y is an extension of Girard's interpretation of intuitionistic logic into linear logic presented in Girard (1987) and developed at the term level in Maraist *et al.* (1999). In this paper we treat this translation semantically using a category of coalgebras, and our first main result shows that the solutions to recursive domain equations from PILL_Y can be given a coalgebra structure satisfying universal conditions, as in Freyd's work on algebraically compact categories. Using this result, we can extend the Girard translation to recursive types.

The example of the model of admissible pers is particularly interesting for two reasons: here the interpretation of FPC can be extended to an interpretation of PolyFPC, an extension of FPC with polymorphism, and this model can be shown to be computationally adequate. The model is to the author's knowledge the first denotational model for the combination of parametric polymorphism, recursive terms and recursive types with a non-linear language. For many readers the construction of this model may be the main result of the paper, but the earlier abstract analysis is needed to show that it models recursive types. The computational adequacy result of the admissible per model also implies computational adequacy for the interpretation of FPC into PILL_Y.

The adequate model of PolyFPC may be used to derive consequences of parametricity, such as modularity proofs, up to ground contextual equivalence along the lines of the proofs of Pitts (2005), but using denotational methods. The model is also interesting because of the mix of parametricity and partiality, a combination that, as earlier research has shown, requires an alternative formulation of parametricity, such as the one suggested in Johann and Voigtländer (2004). This paper describes the resulting parametricity principle derivable from the model, and sketches how the parametric reasoning in the model can be lifted to a logic for parametricity for PolyFPC.

The research presented in this paper also reveals what appears to be a limitation of PILL_Y as a meta language for domain theory and parametric polymorphism, as I have not been able to extend the interpretation of FPC into PILL_Y to PolyFPC, so I have only given the interpretation of full PolyFPC in the specific case of the per-model. It would be an unfortunate setback if no such extension were possible since one would expect that such a meta language would be able to express most constructions of domain theory plus polymorphism.

A related paper is Abadi and Plotkin (1990), where a model of polymorphism and recursion is constructed using admissible pers (as here) satisfying a uniformity property as well as various other properties ensuring that recursive types may be constructed as in domain theory. The main differences between that paper and the current paper is that our model is parametric, and the recursive types in our model are constructed using parametricity.

The paper is organised as follows. Section 2 recalls the language PILL$_Y$ and the theory of models for it, in particular, the per-model. The language PolyFPC is defined in Section 3. Section 4 first recalls recursive domain equations and Freyd's theory of algebraically compact categories since we need to introduce vocabulary and recall results needed for later developments. Using the introduced vocabulary, we present a general notion of FPC model suitable for our purposes and prove soundness of the interpretation of FPC in these. Section 5 introduces the general construction of models of FPC from models of PILL$_Y$. Precisely, if $\mathbf{C}$ with the comonad ! is a parametric PILL$_Y$-model, we prove that the co-Eilenberg–Moore category is an FPC model. In fact, what we prove is more general, in that we show how an algebraic compactness property for a category $\mathbf{C}$ with a given comonad induces an algebraic compactness result for a certain collection of recursive domain equations in the Kleisli category for the derived monad on the co-Eilenberg–Moore category.

Section 6 is a short section providing a sketch of the resulting interpretation of FPC into PILL$_Y$, and in Section 7, the specific case of the per-model is treated. Here the two main results are, as mentioned, that the model extends to PolyFPC and that this model is computationally adequate. In Section 7 the interpretation of PolyFPC into the per-model is presented in elementary terms, and it is our hope that even readers skipping the abstract categorical treatment of Section 5 will be able to understand the interpretation and the following proof of its adequacy.

In Section 8, we give a simple example of proving modularity principles in PolyFPC using the per-model, together with a sketch of the parametricity principle for PolyFPC derivable from the model, as advertised. Finally, Section 9 concludes and discusses the limitations of PILL$_Y$ as a meta language for domain theory and polymorphism as mentioned above.

2. Polymorphic intuitionistic/linear lambda calculus

The calculus PILL$_Y$ is a Polymorphic dual Intuitionistic/Linear Lambda calculus with a fixed point combinator denoted Y. In other words, it is the calculus DILL of Barber (1997) extended with polymorphism and fixed points for terms. This section just gives a sketch of the calculus – see Birkedal *et al.* (2006a) and Møgelberg (2005) for full details.

Types of PILL$_Y$ are given by the grammar

$$\sigma, \tau ::= \alpha \mid I \mid \sigma \otimes \tau \mid \sigma \multimap \tau \mid !\sigma \mid \textstyle\prod \alpha.\, \sigma.$$

We use the notation $\alpha_1, \dots, \alpha_n \vdash \sigma : \mathsf{Type}$ to mean that σ is a well-formed type with free type variables among $\alpha_1, \dots, \alpha_n$. The grammar for terms is

$$\begin{aligned} t \ ::= \ & x \mid \star \mid Y \mid \lambda^{\circ} x : \sigma.t \mid t\, t' \mid t \otimes t' \mid !t \mid \Lambda\alpha : \mathsf{Type}.\, t \mid t(\sigma) \mid \\ & \mathtt{let}\ x : \sigma \otimes y : \tau\ \mathtt{be}\ t\ \mathtt{in}\ t' \mid \mathtt{let}\ !x : \sigma\ \mathtt{be}\ t\ \mathtt{in}\ t' \mid \mathtt{let}\ \star\ \mathtt{be}\ t\ \mathtt{in}\ t'. \end{aligned}$$

Terms have two contexts of term variables: a context of linear variables and a context of intuitionistic variables. The variables of these contexts occur in terms linearly (as in Girard's linear logic (Girard 1987)) and intuitionistically (as in simply typed lambda calculus), respectively. See *loc. cit.* for the term formation rules of the calculus and the equality theory for terms. The term constructor $\lambda^\circ x : \sigma.t$ constructs terms of type $\sigma \multimap \tau$ by abstracting *linear* term variables. Using the Girard encodings, one can define $\sigma \to \tau$ to be $!\sigma \multimap \tau$, and there is a corresponding definable λ-abstraction for *intuitionistic* term variables. Under this convention, the type of the fixed point combinator is $Y : \prod \alpha. (\alpha \to \alpha) \to \alpha$. We refer to the subset of PILL_Y without the fixed point combinator Y as PILL.

One should think of PILL_Y as a type theory for domain theory in combination with polymorphism. Types can be thought of as cpos (complete partial orders) with least elements (also called pointed cpos), interpreting $\otimes$ as the smash product, I as Sierpinski space, $\multimap$ as the domain of continuous bottom preserving (or strict) maps and $!$ as lifting. The encoding of $\to$ using lifting introduces the collection of non-strict continuous maps, so we think of PILL_Y as axiomatising the adjunction

$$\begin{tikzcd} \mathbf{Cppo}_\perp \arrow[r, bend right] \arrow[r, phantom, "\perp"] & \mathbf{Cppo} \arrow[l, bend right] \end{tikzcd}$$

between the categories of pointed cpos with strict continuous maps (on the left-hand side), and all continuous maps (on the right-hand side). In the adjunction, the left adjoint is lifting and the right adjoint is the inclusion. Unfortunately, this interpretation does not immediately extend to polymorphism, so for a model of full PILL_Y, we need to consider admissible pers as in Section 2.2 below, but it is still good to keep the intuition from domain theory in mind.

When Plotkin introduced PILL_Y and suggested it as a type theory for domain theory and polymorphism, he also showed how to encode recursive types using parametric polymorphism. The logic LAPL (Linear Abadi & Plotkin Logic) was introduced in Birkedal *et al.* (2006a). In this logic, the parametricity principle for PILL_Y can be formulated and Plotkin's encodings of recursive types can be verified as consequences of parametricity.

2.1. *PILL_Y-models*

The most general formulation of models of PILL_Y uses fibred category theory, but here we will just consider a large class of PILL_Y-models, which, with the exception of the syntactic models, includes most important models, since the theory of the following sections is much simpler in this case; we will treat syntactic models in Section 6.

A linear category (Benton *et al.* 1992) is a symmetric monoidal closed category $\mathbf{C}$ with a symmetric monoidal comonad $!$ and a given commutative comonoid structure on each object of the form $!A$, such that the maps of the comonoid structure $e_A : !A \multimap I$, $d_A : !A \multimap !A \otimes !A$ are maps of coalgebras from the free coalgebra structure on A to natural structures on I and $!A \otimes !A$. See *loc. cit.* or the more recent Maneggia (2004) and Møgelberg (2005) for details of the definition. As we have already done above, we shall write $f : \sigma \multimap \tau$ for morphisms in $\mathbf{C}$. We say that $\mathbf{C}$ models polymorphism if any functor $F : |\mathbf{C}|^{n+1} \to \mathbf{C}$, where $|\mathbf{C}|$ denotes the objects of $\mathbf{C}$ considered as a discrete category, has a right Kan extension (Mac Lane 1971) $\mathrm{RK}_\pi(F)$ along the projection $\pi : |\mathbf{C}|^{n+1} \to |\mathbf{C}|^n$, satisfying the Beck–Chevalley condtion: if $G : |\mathbf{C}|^m \to |\mathbf{C}|^n$, then the canonical natural transformation

$$\mathrm{RK}_\pi(F) \circ G \to \mathrm{RK}_\pi(F \circ (G \times id_{|\mathbf{C}|}))$$

(constructed as in Jacobs (1999, Section 1.9)) is an isomorphism. This notion of a model of polymorphism was used in Robinson and Rosolini (1994), and further details can also be found in Birkedal and Møgelberg (2005).

A model of PILL is a linear category that models polymorphism, and a model of PILL_Y (or a PILL_Y-model) is a model of PILL with a term modelling the fixed point combinator Y. Given a PILL_Y model $(\mathbf{C}, !)$, types of PILL_Y with n free type variables are modelled as functors $|\mathbf{C}|^n \to \mathbf{C}$ (or equivalently maps $|\mathbf{C}|^n \to |\mathbf{C}|$) by modelling $\vec{\alpha} \vdash \alpha_i$ as the i'th projection, $\otimes, I, \multimap$ using the symmetric monoidal structure, ! using the comonad and polymorphism using the Kan extensions.

When we refer to a parametric model of PILL_Y, we shall mean a PILL_Y-model that extends to a parametric LAPL-structure in the sense of Birkedal *et al.* (2008). Parametric LAPL-structures are models of the logic LAPL in which the parametricity principle holds, and in *loc. cit.* it is shown how to solve recursive domain equations in parametric PILL_Y models. Section 4.1 recalls this result in more detail.

2.2. *A per-model*

This section recalls the parametric PILL_Y-model constructed in Birkedal *et al.* (2007), which should be referred to for details. The model is a variant of the parametric per-model for second-order lambda calculus restricted to a notion of admissible pers to encompass fixed points.

A reflexive domain is a pointed ω-complete partial order D such that the domain $[D \to D]$ of continuous maps from D to D is a retract of D in the sense that there exist continuous maps

$$\begin{tikzcd}
{[D \to D]} \arrow[r, bend right, "\Psi"'] & D \arrow[l, bend right, "\Phi"']
\end{tikzcd}$$

such that $\Phi \circ \Psi$ is the identity. We consider Φ, Ψ as part of the given structure. Reflexive domains were first studied by Scott as models of the untyped lambda calculus (Scott 1970; 1976). A reflexive domain D has a combinatory algebra structure with application $x \cdot y$ defined by applying the function corresponding to x under the reflection to y, that is, $x \cdot y = \Phi(x)(y)$. We shall also need a strict continuous pairing function $\langle -, - \rangle : D \times D \to D$ with continuous projections $\pi, \pi' : D \to D$ such that $\pi(\langle x, y \rangle) = x$ and $\pi'(\langle x, y \rangle) = y$. These are definable from the given data using standard constructions as described in van Oosten (2008). A partial equivalence relation, or a per, on D is a symmetric transitive (but not necessarily reflexive) relation. A per R defines an equivalence relation on its *domain* $|R| = \{x \in D \mid R(x, x)\}$, and we can consider the equivalence classes for this equivalence relation $[x]_R = \{y \in D \mid R(x, y)\}$, and collect them in the set

$$D/R = \{[x]_R \mid x \in |R|\}.$$

We shall often denote a per by its set of equivalence classes. A map of pers from R to S is a map $f : D/R \to D/S$ such that there exists an element $e \in D$ *tracking* f in the sense that $f([x]_R) = [e \cdot x]_S$. This forms a category of pers denoted $\mathbf{Per}(D)$. The following intuition may be useful. The equivalence classes of the per can be thought of as the elements of a datatype, and elements of equivalence classes as different representatives of the same data element. A function f of pers is then a map of data elements that is computable in the sense that there exists a computable function tracking f on the level of representatives.

An *admissible per* is a partial equivalence relation R on D that relates $\bot$ to itself and is *chain complete* in the sense that if $(x_n)_{n\in\mathbb{N}}$ and $(y_n)_{n\in\mathbb{N}}$ are increasing chains of elements in D such that $R(x_n, y_n)$ for all n, then $R(\bigsqcup x_n, \bigsqcup y_n)$ also. We define the category $\mathbf{AP}$ to be the full subcategory of $\mathbf{Per}(D)$ on admissible pers on D, and we define the subcategory $\mathbf{AP}_\bot$ to be the category of admissible pers with strict maps of pers, that is, maps satisfying $f([\bot]) = [\bot]$. We write $f : R \multimap S$ to indicate that f is a strict map of admissible pers.

The category $\mathbf{AP}_\bot$ has products

$$R \times S = \{(\langle x, y\rangle, \langle x', y'\rangle) \mid R(x, x') \wedge S(y, y')\},$$

and a symmetric monoidal closed structure with tensor product defined as a quotient of the product such that strict maps from $R \otimes S$ correspond bijectively to bistrict maps out of $R \times S$. The closed structure is defined as

$$R \multimap S = \{(d, e) \mid d \cdot \bot = e \cdot \bot = \bot \wedge \forall x, y \in D.\, R(x, y) \supset S(d \cdot x, e \cdot y)\}.$$

Finally, there is a symmetric monoidal comonad ! on $\mathbf{AP}_\bot$ defined as mapping R to

$$\{(\bot, \bot)\} \cup \{(\langle \imath, x\rangle, \langle \imath, y\rangle) \mid R(x, y)\}$$

where $\imath = \Psi(id_D)$. This data defines a linear category structure on $\mathbf{AP}_\bot$. The coKleisli category for ! is isomorphic to $\mathbf{AP}$.

Following the intuition for pers given above, a relation on pers R, S is a subset of $D/R \times D/S$. An admissible relation on admissible pers is a subset A of $D/R \times D/S$ containing $([\bot]_R, [\bot]_S)$ that is chain complete in the sense that if $(x_n)_{n\in\mathbb{N}}$ and $(y_n)_{n\in\mathbb{N}}$ are increasing chains of elements in D such that $R(x_n, x_n)$ and $S(y_n, y_n)$ for all n, then if for all n we have $([x_n]_R, [y_n]_S) \in A$, we have $([\bigsqcup x_n]_R, [\bigsqcup y_n]_S) \in A$ also. Admissible relations correspond bijectively to regular subobjects of $R \times S$ in $\mathbf{AP}_\bot$. We write $A : \mathrm{AdmRel}_{\mathbf{AP}_\bot}(R, S)$ to indicate that A is an admissible relation from R to S.

The collection of all admissible relations on admissible pers form a category, denoted $\mathbf{AdmRel}_{\mathbf{AP}_\bot}$, whose morphisms from $A : \mathrm{AdmRel}_{\mathbf{AP}_\bot}(R, S)$ to $B : \mathrm{AdmRel}_{\mathbf{AP}_\bot}(R', S')$ are pairs of morphisms $(f : R \multimap R', g : S \multimap S')$ in $\mathbf{AP}_\bot$ that maps related elements to related elements, that is, if $([x]_R, [y]_S) \in A$, then $(f([x]_R), g([y]_S)) \in A'$. The linear category structure on $\mathbf{AP}_\bot$ can be extended to $\mathbf{AdmRel}_{\mathbf{AP}_\bot}$. For example, if $A : \mathrm{AdmRel}_{\mathbf{AP}_\bot}(R, S)$, then $A : \mathrm{AdmRel}_{\mathbf{AP}_\bot}(!R, !S)$ is the relation relating $[\bot]_R$ to $[\bot]_S$ and relating $[\langle \imath, x\rangle]_{!R}$ to $[\langle \imath, x\rangle]_{!S}$ if and only if $A([x]_R, [y]_S))$. There is a reflexive graph of categories

$$\begin{tikzcd} \mathbf{AdmRel}_{\mathbf{AP}_\bot} \arrow[r, shift left=2] \arrow[r, shift right=2] & \mathbf{AP}_\bot \arrow[l] \end{tikzcd} \qquad (1)$$

where the two functors from left to right map a relation to its domain and codomain, respectively, and the last functor maps a per to the identity relation on the per. The functors in (1) commute with the linear category structure.

The parametric variant of the per-model models an open type σ with n free type variables as a pair $(\llbracket\sigma\rrbracket^r, \llbracket\sigma\rrbracket^p)$, where $\llbracket\sigma\rrbracket^p : |\mathbf{AP}|^n \to |\mathbf{AP}|$ is a map and $\llbracket\sigma\rrbracket^r$ is a map taking an n-vector of admissible relations $(A_i : \mathrm{AdmRel}_{\mathbf{AP}_\perp}(R_i, S_i))_{i\leqslant n}$ and producing an admissible relation

$$\llbracket\sigma\rrbracket^r(\vec{A}) : \mathrm{AdmRel}_{\mathbf{AP}_\perp}(\llbracket\sigma\rrbracket^p(\vec{R}), \llbracket\sigma\rrbracket^p(\vec{S}))$$

satisfying $\llbracket\sigma\rrbracket^r(eq_{R_1}, \ldots, eq_{R_n}) = eq_{\llbracket\sigma\rrbracket^p(\vec{R})}$. Most PILL$_Y$ type constructors are interpreted using the structure described above. For example, ! is interpreted using the comonads defined as above on admissible relations and admissible pers

$$\llbracket !\sigma\rrbracket^p(\vec{R}) = !\llbracket\sigma\rrbracket^p(\vec{R})$$
$$\llbracket !\sigma\rrbracket^r(\vec{A}) = !\llbracket\sigma\rrbracket^r(\vec{A}).$$

Polymorphism is modelled using intersections of pers and relations

$$\begin{aligned}\llbracket\textstyle\prod\alpha.\,\sigma\rrbracket^p(\vec{R}) = \{(x,y) \mid \forall S : |\mathbf{AP}_\perp|.\ \llbracket\sigma\rrbracket^p(\vec{R}, S)(x,y) \wedge \\ \forall S, S' : |\mathbf{AP}|.\,\forall A : \mathrm{AdmRel}_{\mathbf{AP}_\perp}(S, S').\ \llbracket\sigma\rrbracket^r(\vec{eq}_{\vec{R}}, A)([x],[y])\}\\ \llbracket\textstyle\prod\alpha.\,\sigma\rrbracket^r(\vec{A}) = \{([x],[y]) \mid \forall S, S' : |\mathbf{AP}_\perp|.\,\forall A : \mathrm{AdmRel}_{\mathbf{AP}_\perp}(S, S').\ \llbracket\sigma\rrbracket^r(\vec{A}, A)([x],[y])\}\end{aligned}$$

For further details, see Birkedal *et al.* (2007) and Møgelberg (2005).

To see the parametric per-model as an instance of the definition of Section 2.1, one must view this definition as interpreted in a realisability topos, as we now briefly describe. Since D with the application defined as above is a combinatory algebra, one can form the realisability topos RT(D) as in Hyland *et al.* (1980). The categories **AP** and **AP**$_\perp$ are the externalisations of internal categories in RT(D), by which we mean precisely that there exist internal categories in RT(D) (see, for example, Jacobs (1999, Chapter 7) for a textbook exposition of the theory of internal categories) such that **AP** and **AP**$_\perp$ are isomorphic to the categories obtained by applying the global sections functor $\Gamma :$ RT$(D) \to$ **Set** to the internal categories. The construction of these internal categories is similar to the construction of the category of pers over $\mathbb{N}$ as an internal category in the effective topos (see Hyland (1988); see also Jacobs (1999, Example 7.1.3)). Similarly, $\mathbf{AdmRel}_{\mathbf{AP}_\perp}$ is the externalisation of an internal category in RT(D). We shall also write **AP**, **AP**$_\perp$ and $\mathbf{AdmRel}_{\mathbf{AP}_\perp}$ for the internal categories, relying on context to identify whether we are referring to the internal or external categories.

In the theory of internal categories, one can talk about internal functors and internal natural transformations, and use this to formulate notions such as an internal linear category. All the structure on the categories mentioned above is externalisation of internal structure, in the sense that there exist, for example, internal linear category structures on **AP**$_\perp$ and $\mathbf{AdmRel}_{\mathbf{AP}_\perp}$ such that the linear category structures described above arise by applying the global sections functor to the internal structure. So there exists, for example, an internal functor such that the lifting comonad ! on **AP**$_\perp$ described above is the global sections functor applied to the internal functor. Also the functors of diagram (1) are externalisations of internal functors.

The viewpoint of **AP**$_\perp$ and $\mathbf{AdmRel}_{\mathbf{AP}_\perp}$ as internal categories in RT(D) is important because these categories do not model polymorphism when seen as categories in the usual sense (there are simply too many functors and natural transformations in this world), but they do if one restricts to internal functors and internal natural transformations. One can either do this by simply reading the definition as if it refers purely to internal functors and internal natural transformations, or one can use the fact that $\mathrm{RT}(D)$ is a topos to interpret the whole definition of PILL_Y-model in the internal logic of $\mathrm{RT}(D)$ (see, for example, Jacobs (1999), Lambek and Scott (1986) and Johnstone (2002) for the internal logic of a topos). $\mathbf{AP}_\perp$ and $\mathbf{AdmRel}_{\mathbf{AP}_\perp}$ model polymorphism in both of these senses.

Consider the category $\mathrm{RT}(D)^G$ of reflexive graphs over $\mathrm{RT}(D)$, that is, the category that has as objects reflexive graphs in $\mathrm{RT}(D)$ and as morphisms pairs of morphisms making the obvious diagrams commute. It is a well-known fact (used, for example, in Robinson and Rosolini (1994)) that internal categories in $\mathrm{RT}(D)^G$ correspond to diagrams of internal categories in $\mathrm{RT}(D)$, and similarly for functors and natural transformations. As a consequence, diagram (1) describes an internal linear category in $\mathrm{RT}(D)^G$. Diagram (1) is a model of PILL_Y in the sense of Section 2.1 when this definition is interpreted in the internal logic of $\mathrm{RT}(D)^G$, and, in fact, this model is parametric (as proved in Birkedal *et al.* (2007)).

Note that even though the definition of PILL_Y-model had to be understood in an internal sense, the interpretation of PILL_Y that we obtain from this model (as sketched above) is an interpretation in the usual (external) sense.

3. Polymorphic FPC

In this section we present the language PolyFPC, which is an extension of the language FPC, which was first defined by Plotkin (see Plotkin (1985); see also Fiore (1996)), with full impredicative polymorphism. Note that there are several ways of constructing such an extension, since in the definition of the operational semantics one has to choose what to do when a polymorphic lambda abstraction $\Lambda\alpha.\,t$ is met: one can either stop evaluation or continue evaluating t. In this paper, we will only consider the first of these forms of polymorphism.

In later sections we will show how to interpret FPC into any parametric PILL_Y-model of the form of Section 2.1, and how to interpret PolyFPC into the per-model sketched in Section 2.2.

The language PolyFPC has polymorphism and general (nested) recursive types, and therefore types in the languages may have free type variables (as in PILL_Y). Types are formed using the grammar

$$\sigma, \tau ::= \alpha \mid 1 \mid \sigma + \tau \mid \sigma \times \tau \mid \sigma \to \tau \mid \mathrm{rec}\ \alpha.\,\sigma \mid \textstyle\prod \alpha.\,\sigma.$$

As usual, the constructions $\prod \alpha.\,\sigma$ and $\mathrm{rec}\ \alpha.\,\sigma$ bind the type variable α, and, as in PILL_Y, we use the notation $\vec{\alpha} \vdash \sigma$ to mean that σ is a well-formed type with free type variables among $\vec{\alpha}$. The grammar for PolyFPC terms is

$$\begin{aligned} t ::= x \mid \star \mid \mathtt{inl}\ t \mid \mathtt{inr}\ t \mid \mathtt{case}\ t\ \mathtt{of}\ \mathtt{inl}\ x.\,t'\ \mathtt{of}\ \mathtt{inr}\ x.\,t'' \mid \langle t, t' \rangle \mid \\ \pi_1(t) \mid \pi_2(t) \mid \lambda x : \sigma.\,t \mid t(t') \mid \mathtt{intro}\ t \mid \mathtt{elim}\ t \mid \Lambda\alpha.\,t \mid t(\tau). \end{aligned}$$

The typing rules for PolyFPC are listed in Figure 1.

$$\frac{}{\vec{\alpha} \mid x_1 : \sigma_1, \ldots, x_n : \sigma_n \vdash x_i : \sigma_i} \qquad \frac{}{\vec{\alpha} \mid \vec{x} : \vec{\sigma} \vdash \star : 1} \qquad \frac{\vec{\alpha} \mid \vec{x} : \sigma \vdash t : \tau}{\vec{\alpha} \mid \vec{x} : \sigma \vdash \mathtt{inl}\, t : \tau + \tau'}$$

$$\frac{\vec{\alpha} \mid \vec{x} : \sigma \vdash t : \tau'}{\vec{\alpha} \mid \vec{x} : \sigma \vdash \mathtt{inr}\, t : \tau + \tau'} \qquad \frac{\vec{\alpha} \mid \vec{x} : \vec{\sigma} \vdash t : \tau + \tau' \quad \vec{\alpha} \mid \vec{x} : \vec{\sigma}, x : \tau \vdash t' : \omega \quad \vec{\alpha} \mid \vec{x} : \vec{\sigma}, x : \tau' \vdash t'' : \omega}{\vec{\alpha} \mid \vec{x} : \vec{\sigma} \vdash \mathtt{case}\, t\, \mathtt{of}\, \mathtt{inl}\, x. t'\, \mathtt{of}\, \mathtt{inr}\, x. t'' : \omega}$$

$$\frac{\vec{\alpha} \mid \vec{x} : \vec{\sigma} \vdash t : \tau \quad \vec{\alpha} \mid \vec{x} : \vec{\sigma} \vdash t' : \tau'}{\vec{\alpha} \mid \vec{x} : \vec{\sigma} \vdash \langle t, t' \rangle : \tau \times \tau'} \qquad \frac{\vec{\alpha} \mid \vec{x} : \vec{\sigma} \vdash t : \tau \times \tau'}{\vec{\alpha} \mid \vec{x} : \vec{\sigma} \vdash \pi_1(t) : \tau} \qquad \frac{\vec{\alpha} \mid \vec{x} : \vec{\sigma} \vdash t : \tau \times \tau'}{\vec{\alpha} \mid \vec{x} : \vec{\sigma} \vdash \pi_2(t) : \tau'}$$

$$\frac{\vec{\alpha} \mid \vec{x} : \vec{\sigma}, x : \tau \vdash t : \tau'}{\vec{\alpha} \mid \vec{x} : \vec{\sigma} \vdash \lambda x : \tau. t : \tau \to \tau'} \qquad \frac{\vec{\alpha} \mid \vec{x} : \vec{\sigma} \vdash t : \tau \to \tau' \quad \vec{\alpha} \mid \vec{x} : \vec{\sigma} \vdash t' : \tau}{\vec{\alpha} \mid \vec{x} : \vec{\sigma} \vdash t(t') : \tau'}$$

$$\frac{\vec{\alpha} \mid \vec{x} : \vec{\sigma} \vdash t : \tau[\mathrm{rec}\ \beta. \tau / \beta]}{\vec{\alpha} \mid \vec{x} : \vec{\sigma} \vdash \mathtt{intro}\, t : \mathrm{rec}\ \beta. \tau} \qquad \frac{\vec{\alpha} \mid \vec{x} : \vec{\sigma} \vdash t : \mathrm{rec}\ \beta. \tau}{\vec{\alpha} \mid \vec{x} : \vec{\sigma} \vdash \mathtt{elim}\, t : \tau[\mathrm{rec}\ \beta. \tau / \beta]}$$

$$\frac{\vec{\alpha}, \beta \mid \vec{x} : \vec{\sigma} \vdash t : \tau}{\vec{\alpha} \mid \vec{x} : \vec{\sigma} \vdash \Lambda \beta. t : \prod \beta. \tau}\ \vec{\alpha} \vdash \vec{\sigma} \qquad \frac{\vec{\alpha} \mid \vec{x} : \vec{\sigma} \vdash t : \prod \beta. \tau \quad \vec{\alpha} \vdash \tau'}{\vec{\alpha} \mid \vec{x} : \vec{\sigma} \vdash t(\tau') : \tau[\tau'/\beta]}$$

Fig. 1. Typing rules for PolyFPC

In the following, we shall use the word 'programs', to mean closed typable terms of closed type. The language PolyFPC is equipped with a call-by-value operational semantics. Formally, the operational semantics is a relation $\Downarrow$ relating programs to values, by which we mean programs following the grammar

$$v \ ::= \star \mid \mathtt{inl}\, v \mid \mathtt{inr}\, v \mid \langle v, v' \rangle \mid \lambda x : \sigma. t \mid \Lambda \alpha. t \mid \mathtt{intro}\, v.$$

The operational semantics is given by the rules listed in Figure 2.

The sublanguage FPC is the part of PolyFPC not mentioning polymorphism, that is: the grammar for types is the same as the grammar for PolyFPC except for $\prod \alpha. \sigma$; the grammar for FPC terms does not include the $\Lambda \alpha. t$ and $t(\tau)$; and the formation rules for FPC are as in Figure 1 except for the last two rules. Similarly, the operational semantics for FPC is defined by the obvious restriction.

Remark 3.1. The languages FPC and PolyFPC could easily be extended with recursion on the level of terms. For example, one could add the term formation rule

$$\frac{\vec{\alpha} \mid \vec{x} : \vec{\sigma}, f : \tau \to \tau', x : \tau \vdash t : \tau' \quad \vec{\alpha} \mid \vec{x} : \vec{\sigma} \vdash t' : \tau}{\vec{\alpha} \mid \vec{x} : \vec{\sigma} \vdash \mathtt{letrec}\, f x = t\ \mathtt{in}\, f\ t' : \tau'}$$

$$v \Downarrow v \qquad \frac{e \Downarrow v}{\mathtt{inl}\, e \Downarrow \mathtt{inl}\, v} \qquad \frac{e \Downarrow v}{\mathtt{inr}\, e \Downarrow \mathtt{inr}\, v}$$

$$\frac{e \Downarrow \mathtt{inl}\, v \qquad e'[v/x] \Downarrow v'}{\mathtt{case}\, e\, \mathtt{of}\, \mathtt{inl}\, x.\, e'\, \mathtt{of}\, \mathtt{inr}\, x.\, e'' \Downarrow v'} \qquad \frac{e \Downarrow \mathtt{inr}\, v \qquad e''[v/x] \Downarrow v'}{\mathtt{case}\, e\, \mathtt{of}\, \mathtt{inl}\, x.\, e'\, \mathtt{of}\, \mathtt{inr}\, x.\, e'' \Downarrow v'}$$

$$\frac{e \Downarrow v \qquad e' \Downarrow v'}{\langle e, e' \rangle \Downarrow \langle v, v' \rangle} \qquad \frac{e \Downarrow \langle v, v' \rangle}{\pi_1(e) \Downarrow v} \qquad \frac{e \Downarrow \langle v, v' \rangle}{\pi_2(e) \Downarrow v'} \qquad \frac{e \Downarrow \lambda x.\, e'' \qquad e' \Downarrow v \qquad e''[v/x] \Downarrow v'}{e(e') \Downarrow v'}$$

$$\frac{e \Downarrow \mathtt{intro}\, v}{\mathtt{elim}\, e \Downarrow v} \qquad \frac{e \Downarrow v}{\mathtt{intro}\, e \Downarrow \mathtt{intro}\, v} \qquad \frac{t \Downarrow \Lambda\alpha.\, t' \qquad t'[\tau/\alpha] \Downarrow v}{t(\tau) \Downarrow v}$$

Fig. 2. Operational semantics for PolyFPC

and add the rule

$$\frac{e' \Downarrow v' \qquad e[\lambda x : \tau.\, \mathtt{letrec}\, f x' = e\, \mathtt{in}\, f\ x/f, v'/x'] \Downarrow v}{\mathtt{letrec}\, f x' = e\, \mathtt{in}\, f\ e' \Downarrow v}$$

to the operational semantics. However, it is standard that recursion on the level of terms can be encoded using recursion on the level of types, by defining for each type σ a fixed point combinator $fix_\sigma : (\sigma \to \sigma) \to \sigma$ as $\lambda f : \sigma \to \sigma. k(\mathtt{intro}\,(k))$ where $k = \lambda x : \mathrm{rec}\ \alpha.\,(\alpha \to \sigma).\, f(\mathtt{elim}\,(x)(x))$ satisfying $f(fix_\sigma f) \Downarrow v$ if and only if $fix_\sigma f \Downarrow v$ for all programs f and values v.

4. Recursive domain equations

This section recalls Freyd's theory (Freyd 1990b; 1990a; 1991) of algebraically compact categories and introduces the vocabulary we will need later in the paper. In the following, we will use $\mathbf{C}$ to denote a category, which is usually assumed to be cartesian closed or symmetric monoidal closed such that basic type constructors such as $\times$ or $\otimes$ and $\to$ or $\multimap$ exists giving us an interesting collection of domain equations.

Syntactically, a recursive type equation is given by a type expression σ with a free variable α, and a solution is a type τ such that $\sigma[\tau/\alpha] \cong \tau$. Usually, one is not just interested in any solution, but rather a solution satisfying a universal condition. For the formulation of such a universal condition, one considers positive and negative occurrences of a variable in a type expression: α occurs positively in the type expression α and the introduction of an arrow $\to$ reverses parity on the left of the arrow and preserves it on the right. For example, α occurs only positively in the types $\sigma = \alpha + 1$ and $\sigma = (\alpha \to \tau) \to \tau$ if τ is a closed type, but it occurs negatively in $(\tau \to \alpha) \to \tau$. If α occurs only positively in σ, the interpretation of σ induces a functor $\mathbf{C} \to \mathbf{C}$, and we can ask for initial algebras or final coalgebras for this functor.

For the more general case of both positive and negative occurrences of α in σ (such as $\sigma = (\alpha \to \alpha) + 1$), one can split the occurrences of α in σ into positive and negative, and obtain a type $\alpha, \beta \vdash \sigma$ where α occurs only negatively and β only positively. The interpretation of such a type induces a functor $F : \mathbf{C}^{\mathrm{op}} \times \mathbf{C} \to \mathbf{C}$. We cannot talk about initial algebras or final coalgebras for F as it is not an endofunctor, but we can symmetrise F to the functor $\breve{F} : \mathbf{C}^{\mathrm{op}} \times \mathbf{C} \to \mathbf{C}^{\mathrm{op}} \times \mathbf{C}$ defined as

$$\breve{F}(X,Y) = (F(Y,X), F(X,Y)),$$

and consider initial algebras and final coalgebras for $\breve{F}$. Writing out the category of algebras for $\breve{F}$, we arrive at the category of dialgebras for F: a dialgebra for F is a quadruple (X, X', f, f') where X, X' are objects of $\mathbf{C}$ and $f : F(X', X) \to X$ and $f' : X' \to F(X, X')$ are morphisms. A morphism of dialgebras from (X, X', f, f') to (Y, Y', g, g') is a pair of maps $h : X \to Y$ and $h' : Y' \to X'$ such that the diagrams

$$\begin{tikzcd} F(X',X) \arrow[r, "f"] \arrow[d, "F(h',h)"'] & X \arrow[d, "h"] \\ F(Y',Y) \arrow[r, "g"] & Y \end{tikzcd} \quad \text{and} \quad \begin{tikzcd} Y' \arrow[r, "g'"] \arrow[d, "h'"'] & F(Y,Y') \arrow[d, "F(h,h')"] \\ X' \arrow[r, "f'"] & F(X,X') \end{tikzcd}$$

commute. An initial dialgebra is an initial object in the category of dialgebras.

Initial dialgebras generalise initial algebras and final coalgebras because, given a functor $F : \mathbf{C} \to \mathbf{C}$, we can consider the composition of F with the projection $\mathbf{C}^{\mathrm{op}} \times \mathbf{C} \to \mathbf{C}$, and (X, X', f, f') is an initial dialgebra for this functor if and only if f is an initial algebra for F and f' is a final coalgebra for F. Freyd showed in Freyd (1990b; 1990a; 1991) that if a category $\mathbf{C}$ is algebraically compact, that is, all endofunctors have initial algebras and coalgebras and, moreover, these coincide in the sense that the inverse of an initial algebra is a final coalgebra, then initial dialgebras exist on the diagonal, that is, there exist initial dialgebras of the form (X, X, f, f^{-1}) for some isomorphism f. For a precise formulation of this result, see Theorem 4.5 below.

To solve general nested recursive type equations, we need to consider type equations with variables. Syntactically, these are given by types $\vec{\alpha}, \beta \vdash \sigma(\vec{\alpha}, \beta)$, and a solution is a type $\vec{\alpha} \vdash \tau$ such that $\sigma(\vec{\alpha}, \tau) \cong \tau$. The type equations induce functors $F : (\mathbf{C}^{\mathrm{op}} \times \mathbf{C})^{n+1} \to \mathbf{C}$, which can be symmetrised to

$$\breve{F} : (\mathbf{C}^{\mathrm{op}} \times \mathbf{C})^{n+1} \to \mathbf{C}^{\mathrm{op}} \times \mathbf{C}$$

defined by

$$\breve{F}(A_1, B_1, \ldots A_{n+1}, B_{n+1}) = (F(B_1, A_1, \ldots B_{n+1}, A_{n+1}), F(A_1, B_1, \ldots A_{n+1}, B_{n+1})).$$

Below, we shall also use the notation $F^{\diamond} : (\mathbf{C}^{\mathrm{op}} \times \mathbf{C})^{n+1} \to \mathbf{C}^{\mathrm{op}}$ for the functor that maps $(A_1, B_1, \ldots A_{n+1}, B_{n+1})$ to $F(B_1, A_1, \ldots B_{n+1}, A_{n+1})$, for example, $\breve{F} = \langle F^{\diamond}, F \rangle$. Similarly, if $F, F' : (\mathbf{C}^{\mathrm{op}} \times \mathbf{C})^{n+1} \to \mathbf{C}$ are functors and h is a natural transformation from F to F', we use the notation $h^{\diamond}$ for the natural transformation from $F'^{\diamond}$ to $F^{\diamond}$ given by the components

$$h^{\diamond}_{(A_1, B_1, \ldots A_{n+1}, B_{n+1})} = h_{(B_1, A_1, \ldots B_{n+1}, A_{n+1})}.$$

The next definition states what we mean by being able to solve recursive domain equations in a category. The definition is given with respect to a class functors $\mathscr{F}$, which we think of as our class of domain equations.

Definition 4.1. A class $\mathscr{F}$ of functors of the form $F\colon (\mathbf{C}^{\mathrm{op}}\times\mathbf{C})^{n+1}\to\mathbf{C}$ is a *collection of recursive type equations* if it contains all projections and is closed under composition in the sense that if $F\colon (\mathbf{C}^{\mathrm{op}}\times\mathbf{C})^{n}\to\mathbf{C}$ is in $\mathscr{F}$ and, similarly, $G_i\colon (\mathbf{C}^{\mathrm{op}}\times\mathbf{C})^{m}\to\mathbf{C}$ and $G_i'\colon (\mathbf{C}^{\mathrm{op}}\times\mathbf{C})^{m}\to\mathbf{C}$ for $i=1,\dots n$, then $F\circ\langle\langle {G_1'}^{\Diamond},G_1\rangle,\dots,\langle {G_n'}^{\Diamond},G_n\rangle\rangle$ is in $\mathscr{F}$ also.

We say that $\mathbf{C}$ has solutions to the collection of recursive domain equations $\mathscr{F}$ if for any functor $F\colon (\mathbf{C}^{\mathrm{op}}\times\mathbf{C})^{n+1}\to\mathbf{C}$ in $\mathscr{F}$ there exists a functor $\mathrm{Fix}F\colon (\mathbf{C}^{\mathrm{op}}\times\mathbf{C})^{n}\to\mathbf{C}$ in $\mathscr{F}$ such that

$$F\circ\langle id_{(\mathbf{C}^{\mathrm{op}}\times\mathbf{C})^n},\breve{\mathrm{Fix}F}\rangle\cong\mathrm{Fix}F,$$

and, furthermore, $\mathrm{Fix}F$ satisfies the initial dialgebra condition, meaning that for any pair of functors

$$G,G'\colon (\mathbf{C}^{\mathrm{op}}\times\mathbf{C})^{n}\to\mathbf{C}$$

in $\mathscr{F}$ and natural transformations

$$g\colon F\circ\langle id,\langle G'^{\Diamond},G\rangle\rangle\to G \quad\text{and}\quad g'\colon G'\to F\circ\langle id,\langle G^{\Diamond},G'\rangle\rangle,$$

there exist unique natural transformations $h\colon \mathrm{Fix}F\to G$, $h'\colon G'\to\mathrm{Fix}F$ making the diagrams

$$\begin{tikzcd}
F\circ\langle id,\breve{\mathrm{Fix}F}\rangle \arrow[r, "\cong"] \arrow[d, "{F(id,h'^{\Diamond},h)}"'] & \mathrm{Fix}F \arrow[d, "h"] \\
F\circ\langle id,\langle G'^{\Diamond},G\rangle\rangle \arrow[r, "g"] & G
\end{tikzcd}
\quad\text{and}\quad
\begin{tikzcd}
G' \arrow[r, "g'"] \arrow[d, "h'"'] & F\circ\langle id,\langle G^{\Diamond},G'\rangle\rangle \arrow[d, "{F(id,h^{\Diamond},h')}"] \\
\mathrm{Fix}F \arrow[r, "\cong"] & F\circ\langle id,\breve{\mathrm{Fix}F}\rangle
\end{tikzcd}$$

commute. The functor $\mathrm{Fix}F$ is said to be the solution to the domain equation given by F.

The most famous example of a category with solutions to recursive domain equations is the category of pointed cpos and strict continuous functions, where the collection of equations is given by the locally continuous functors.

We now recall two well-known results from the theory of recursive types. Proofs of Lemma 4.2 and Lemma 4.4 can be found in, for example, Fiore's thesis (Fiore 1996).

Lemma 4.2. Taking fixed points commutes with reindexing, that is, if

$$F\colon (\mathbf{C}^{\mathrm{op}}\times\mathbf{C})^{n+1}\to\mathbf{C} \quad\text{and}\quad G_i\colon (\mathbf{C}^{\mathrm{op}}\times\mathbf{C})^{m}\to\mathbf{C}$$

for $i=1,\dots n$, then

$$\mathrm{Fix}(F\circ(\langle\breve{G}_1,\dots,\breve{G}_n\rangle\times id_{((\mathbf{C})^{\mathrm{op}}\times\mathbf{C})}))\cong(\mathrm{Fix}\,F)\circ\langle\breve{G}_1,\dots,\breve{G}_n\rangle.$$

Definition 4.3. A collection of recursive domain equations has a strict choice of solutions if it comes equipped with a choice of solutions for each equation such that the isomorphisms of Lemma 4.2 are identities.

Lemma 4.4. If $F\colon \mathbf{D}\times\mathbf{C}\to\mathbf{C}$ is a functor such that for each object X in $\mathbf{D}$, the functor $F(X,-)\colon \mathbf{C}\to\mathbf{C}$ has an initial algebra $in_X\colon F(X,\mu Y.F(X,Y))\to\mu Y.F(X,Y)$, then the assignment $X \mapsto \mu Y.F(X,Y)$ extends to a functor $\mu Y.F(-,Y) \colon \mathbf{D} \to \mathbf{C}$ mapping $f \colon X \to X'$ in $\mathbf{D}$ to the unique map $\mu Y.F(f,Y)$ making

$$\begin{tikzcd}
F(X, \mu Y.F(X,Y)) \arrow[rr, "in_X"] \arrow[d, "F(id_X, \mu Y.F(f,Y))"'] & & \mu Y.F(X,Y) \arrow[d, "\mu Y.F(f,Y)"] \\
F(X, \mu Y.F(X',Y)) \arrow[r, "F(f,id)"] & F(X', \mu Y.F(X',Y)) \arrow[r, "in_{Y'}"] & \mu Y.F(X',Y)
\end{tikzcd}$$

commute.

Finally, we formulate Freyd's theorem in the language of this section. In the theorem, when we refer to functors of the form $F \colon (\mathbf{C}^{\mathrm{op}} \times \mathbf{C})^n \times \mathbf{C} \to \mathbf{C}$ in $\mathscr{F}$ where $\mathscr{F}$ is a collection of recursive domain equations, we shall mean functors F such that $F \circ \pi \colon (\mathbf{C}^{\mathrm{op}} \times \mathbf{C})^{n+1} \to \mathbf{C}$ is in $\mathscr{F}$, where π is the projection $(\mathbf{C}^{\mathrm{op}} \times \mathbf{C})^{n+1} \to (\mathbf{C}^{\mathrm{op}} \times \mathbf{C})^n \times \mathbf{C}$.

Theorem 4.5. A category $\mathbf{C}$ has solutions to a class of domain equations $\mathscr{F}$ if and only if each $F \colon \mathbf{C} \to \mathbf{C}$ in $\mathscr{F}$ has an initial algebra whose inverse is a final coalgebra and, furthermore, for each $F \colon (\mathbf{C}^{\mathrm{op}} \times \mathbf{C})^n \times \mathbf{C} \to \mathbf{C}$ in $\mathscr{F}$, the functor $\mu X.F \colon (\mathbf{C}^{\mathrm{op}} \times \mathbf{C})^n \to \mathbf{C}$ obtained by taking pointwise initial algebras as in Lemma 4.4 is in $\mathscr{F}$.

4.1. *Recursive domain equations in PILL$_Y$*

This subsection reviews results for solutions to recursive domain equations in models of PILL$_Y$, and presents them in the language of Definition 4.1. Thus, in the following, we use $\mathbf{C}$ to denote a model of PILL$_Y$ in the sense of Section 2.1. Following the convention of PILL$_Y$, we write $\multimap$ for maps in $\mathbf{C}$ and reserve $\to$ for maps in the co-Kleisli category, that is, $f \colon \sigma \to \tau$ is shorthand for $f \colon !\sigma \multimap \tau$.

Any PILL$_Y$ type $\alpha \vdash \sigma$ in which α occurs only positively induces a functor $\mathbf{C} \to \mathbf{C}$, or, more generally, any type $\alpha_1, \dots, \alpha_n \vdash \sigma$ induces after a splitting of occurrences of variables into positive and negative a functor $(\mathbf{C}^{\mathrm{op}} \times \mathbf{C})^n \to \mathbf{C}$. More precisely, in Birkedal *et al.* (2006a), a term

$$M_\sigma \colon \textstyle\prod \vec{\alpha}, \vec{\beta}, \vec{\alpha}', \vec{\beta}' : \mathsf{Type}.\, (\vec{\alpha}' \multimap \vec{\alpha}) \to (\vec{\beta} \multimap \vec{\beta}') \to \sigma(\vec{\alpha}, \vec{\beta}) \multimap \sigma(\vec{\alpha}', \vec{\beta}')$$

is defined for each type σ by induction over the structure of σ such that, writing $\sigma(\vec{f}, \vec{g})$ for

$$M_\sigma \vec{\alpha}\, \vec{\beta}\, \vec{\alpha}'\, \vec{\beta}'\, (!\vec{f})\, (!\vec{g}),$$

we have $\sigma(\vec{id}, \vec{id}) = id$ and $\sigma(\vec{f} \circ \vec{f}', \vec{g}' \circ \vec{g}) = \sigma(\vec{f}', \vec{g}') \circ \sigma(\vec{f}, \vec{g})$. This term induces the functorial action of σ. In general, we shall say a functor $F \colon (\mathbf{C}^{\mathrm{op}} \times \mathbf{C})^n \to \mathbf{C}$ is *strong* if there exists a term *in the model* inducing it. For example, all constant functors are strong because for any object X in the model, the constant map to the identity on X is a term in the model. For further details, see Birkedal *et al.* (2008) and Møgelberg (2005).

Theorem 4.6. If $\mathbf{C}$ is a parametric PILL$_Y$ model, it has solutions to the class of recursive domain equations given by strong functors.

Theorem 4.6 is due to Plotkin, who showed how to encode initial algebras and final coalgebras in PILL$_Y$. Because of the fixed point combinator in PILL$_Y$, initial algebras and final coalgebras coincide, so Freyd's theory of algebraically compact categories applies and gives solutions to recursive domain equations. Syntactically, the solutions to recursive type equations produce for any type σ a type rec $\alpha.\,\sigma$ satisfying $\sigma[\text{rec } \alpha.\,\sigma/\alpha] \cong \text{rec } \alpha.\,\sigma$. This syntactic construction commutes with reindexing in the sense that rec $\alpha.\,(\sigma[\vec{\tau}/\vec{\beta}]) = (\text{rec } \alpha.\,\sigma)[\vec{\tau}/\vec{\beta}]$. In PILL$_Y$ models are strict in the sense that it is given with a choice of Kan extensions such that the maps of the Beck–Chevalley condition are all identities, substitution in types is interpreted as composition in the sense that

$$[\![\vec{\beta} \vdash \sigma[\vec{\tau}/\vec{\alpha}]]\!] = [\![\vec{\alpha} \vdash \sigma]\!] \circ \langle [\![\vec{\beta} \vdash \tau_1]\!], \ldots, [\![\vec{\beta} \vdash \tau_n]\!] \rangle,$$

so these PILL$_Y$ models come equipped with a strict choice of solutions to the recursive domain equiations given by strong functors. For a detailed proof of Theorem 4.6, see Birkedal *et al.* (2006a) and Birkedal *et al.* (2008).

4.2. *Models of FPC*

This section defines the notions of FPC model and PolyFPC model that will be used in this paper. In axiomatic domain theory, FPC is usually interpreted in a category **C** where a given collection of subobjects gives rise to a category of partial maps in which the original category **C** forms a full-on-objects subcategory of total maps (see Fiore's thesis (Fiore 1996)). In these models, terms of FPC are interpreted as partial maps, but the category of total maps still plays a key role as the values of FPC are interpreted as total maps, which is a fact used in establishing soundness.

In many cases there exists a monad L on **C** such that the category of partial maps is isomorphic to the Kleisli category $\mathbf{C}_L$. The main example that the reader may know, and should keep in mind, is that of the category **Cpo** of complete partial orders and continuous maps with the lifting monad L. Therefore, the notion of an FPC model given below may be seen as an axiomatisation of the properties of the Kleisli adjunction

$$\begin{tikzcd} \mathbf{Cpo} \arrow[r, bend right] \arrow[r, phantom, "\top"] & \mathbf{Cpo}_L \arrow[l, bend right] \end{tikzcd}$$

Note that $\mathbf{Cpo}_L$ is isomorphic to the category of pointed cpos with strict maps $\mathbf{Cppo}_\perp$, so one often sees FPC interpreted in the latter category.

Our definition of an FPC model differs from that of Fiore by taking the notion of the lifting monad as primitive rather than the notion of partiality. This definition is chosen because we want to construct FPC models from PILL$_Y$ models, and in the definition of a PILL$_Y$ model, the comonad is taken as primitive and could *a priori* be any comonad. The comonad can be used to construct categories with monads. On the other hand, there is no natural notion of partiality in the definition of a PILL$_Y$ model, so we prefer monads for the definition of FPC models.

In the following, if **C** is a category and L is a monad on **C**, we use $i\colon \mathbf{C} \to \mathbf{C}_L$ to denote the left adjoint of the usual adjunction, and use the notation $f : X \to Y$ for maps of **C** and $f : X \rightharpoonup Y$ for maps of the Kleisli category $\mathbf{C}_L$. Even though L can be any monad, it is still a useful intuition that $\mathbf{C}_L$ is a category of partial maps, and, following this intuition, we say that a map $f : X \rightharpoonup Y$ in $\mathbf{C}_L$ is *total* if it is in the image of i.

Recall that if $\mathbf{C}$ is cartesian and L is commutative, the product on $\mathbf{C}$ induces a symmetric monoidal structure on the Kleisli category $\mathbf{C}_L$: on objects, the tensor product is given by the cartesian product on $\mathbf{C}$ and for morphisms $f : X \rightharpoonup Y, g : X' \rightharpoonup Y'$ the tensor product $f \otimes g : X \otimes Y \rightharpoonup X' \otimes Y'$ is given by the composition

$$X \times Y \xrightarrow{f \times g} LX' \times LY' \xrightarrow{dst} L(X' \times Y')$$

where dst is the double strength (see Jacobs (1994) for further details).

Definition 4.7. A cartesian category $\mathbf{C}$ with coproducts and a commutative monad L is:

— an *FPC-model* if:

- The Kleisli category $\mathbf{C}_L$ has Kleisli exponentials, that is, for every object X of $\mathbf{C}$, the composite

$$\mathbf{C} \xrightarrow{X \times (-)} \mathbf{C} \xrightarrow{i} \mathbf{C}_L$$

  has a right adjoint $X \rightharpoonup (-) : (\mathbf{C})_L \to \mathbf{C}$.

- There exists a class of recursive domain equations $\mathscr{F}$ on $\mathbf{C}_L$ containing Kleisli exponentials (see Lemma 4.8 below) and the $\otimes$ induced by the product on $\mathbf{C}$ such that $\mathscr{F}$ has solutions. Moreover, all components of the isomorphisms

$$F \circ \langle id_{(\mathbf{C}^{\mathrm{op}} \times \mathbf{C})^n}, \breve{\mathrm{Fix}} F \rangle \cong \mathrm{Fix} F$$

  must be total, that is, in the image of i.

— a *PolyFPC model* if:

- It is an FPC-model
- It models Kleisli polymorphism in the sense that for any map $\tau : |\mathbf{C}|^{n+1} \to |\mathbf{C}|$ (where we use $|\mathbf{C}|$ to denote the collection of objects of $\mathbf{C}$) the composite $L \circ \tau : |\mathbf{C}|^{n+1} \to \mathbf{C}$ considered as a functor from the discrete category on $|\mathbf{C}|^{n+1}$ has a right Kan extension (Mac Lane 1971) $\mathrm{RK}_\pi(L \circ \tau) : |\mathbf{C}|^n \to \mathbf{C}$ along the projection $\pi : |\mathbf{C}|^{n+1} \to |\mathbf{C}|^n$. The Kan extensions must satisfy the Beck–Chevalley condition: that is, if $\tau : |\mathbf{C}|^{n+1} \to |\mathbf{C}|$ and $\sigma : |\mathbf{C}|^m \to |\mathbf{C}|^n$, the canonical natural transformation

$$\mathrm{RK}_\pi(L \circ \tau) \circ \sigma \to \mathrm{RK}_\pi(L \circ \tau \circ (\sigma \times id_{|\mathbf{C}|}))$$

  (constructed as in Jacobs (1999, Section 1.9)) is an isomorphism.

- The class of domain equations for which we can solve recursive domain equations is closed under Kleisli polymorphism, that is, if $F : ({\mathbf{C}_L}^{\mathrm{op}} \times \mathbf{C}_L)^{n+1} \to \mathbf{C}_L$ is in $\mathscr{F}$, then so is $i \circ \tilde{\prod} F : ({\mathbf{C}_L}^{\mathrm{op}} \times \mathbf{C}_L)^n \to \mathbf{C}_L$ where $\tilde{\prod} F$ is the induced functor defined as in Lemma 4.12 below.

— An FPC model is *strict* if the class of domain equations comes equipped with a strict choice of solutions. A PolyFPC model is strict if it is strict as an FPC model and it comes equipped with a canonical choice of Kan extensions modelling Kleisli polymorphism such that the isomorphisms of the Beck–Chevalley condition are identities.

Lemma 4.8. The Kleisli function space induces a functor

$$(-) \rightharpoonup (=) \colon \mathbf{C}_L{}^{\mathrm{op}} \times \mathbf{C}_L \to \mathbf{C}.$$

Lemma 4.8 is proved by a standard verification.

Lemma 4.9. The correspondence between the maps $X \otimes Y \rightharpoonup Z$ in $\mathbf{C}_L$ and the maps $X \to [Y \rightharpoonup Z]$ in $\mathbf{C}$ is natural in X for total maps.

Lemma 4.9 follows from $(-) \otimes Y$ being adjoint to $Y \rightharpoonup (-)$.

Lemma 4.10. If $(\mathbf{C}, L)$ is an FPC model, then $\mathbf{C}_L$ has coproducts and the tensor product distributes over coproducts, that is, $X \otimes (Y + Z) \cong X \otimes Y + X \otimes Z$. This isomorphism is natural in Y and Z for total maps and in X for partial maps.

Proof. Since $X \times (-) \colon \mathbf{C} \to \mathbf{C}_L$ has a right adjoint $X \rightharpoonup (-)$, it preserves coproducts, and since coproducts in $\mathbf{C}$ and $\mathbf{C}_L$ agree, we get the desired isomorphism. $\square$

In the following, given categories $\mathbf{A}, \mathbf{B}$, we shall write $[\mathbf{A}, \mathbf{B}]$ for the category of functors from $\mathbf{A}$ to $\mathbf{B}$ with natural transformations as morphisms. We shall also write $\mathrm{Nat}_{[\mathbf{A},\mathbf{B}]}(F, G)$ for the collection of natural transformations from F to G when both F, G are functors $\mathbf{A} \to \mathbf{B}$. Finally, we shall use the notation $|\mathbf{C}|$ for both the collection of objects for $\mathbf{C}$ and the discrete category on $|\mathbf{C}|$. Note that maps $|\mathbf{C}| \to |\mathbf{C}|$ are the same as functors $|\mathbf{C}| \to \mathbf{C}$, which are the same as functors $|\mathbf{C}| \to \mathbf{C}_L$.

Lemma 4.11. If $(\mathbf{C}, L)$ is a PolyFPC model, then the mapping $(\tau \colon |\mathbf{C}|^{n+1} \to |\mathbf{C}|) \mapsto (\mathrm{RK}_\pi(L \circ \tau) \colon |\mathbf{C}|^n \to |\mathbf{C}|)$ extends to a functor $[|\mathbf{C}|^{n+1}, \mathbf{C}_L] \to [|\mathbf{C}|^n, \mathbf{C}]$, which is right adjoint to the functor mapping $\sigma \colon |\mathbf{C}|^n \to \mathbf{C}$ to $i \circ \sigma \circ \pi$, where $\pi \colon |\mathbf{C}|^{n+1} \to |\mathbf{C}|^n$ is the projection. In other words, if $\sigma \colon |\mathbf{C}|^n \to |\mathbf{C}|$ and $\tau \colon |\mathbf{C}|^{n+1} \to |\mathbf{C}|$ are maps, then there exists a bijective correspondence

$$\mathrm{Nat}_{[|\mathbf{C}|^{n+1}, \mathbf{C}_L]}(\sigma \circ \pi, \tau) \cong \mathrm{Nat}_{[|\mathbf{C}|^n, \mathbf{C}]}(\sigma, \mathrm{RK}_\pi(L \circ \tau)),$$

which is natural in transformations of total maps in σ and natural transformations of partial maps in τ.

Proof. From the theory of Kan extensions (see Mac Lane(1971, X.3)), we get the bijective correspondence

$$\mathrm{Nat}_{[|\mathbf{C}|^{n+1}, \mathbf{C}]}(\sigma \circ \pi, L \circ \tau) \cong \mathrm{Nat}_{[|\mathbf{C}|^n, \mathbf{C}]}(\sigma, \mathrm{RK}_\pi(L \circ \tau)),$$

which is natural in σ and $L \circ \tau$. The lemma follows from composing this with the isomorphism

$$\mathrm{Nat}_{[|\mathbf{C}|^{n+1}, \mathbf{C}_L]}(i \circ \sigma \circ \pi, \tau) \cong \mathrm{Nat}_{[|\mathbf{C}|^{n+1}, \mathbf{C}]}(\sigma \circ \pi, L \circ \tau)$$

natural in σ and τ. $\square$

Lemma 4.12. Suppose $(\mathbf{C}, L)$ is a PolyFPC model and $F \colon (\mathbf{C}_L{}^{\mathrm{op}} \times \mathbf{C}_L)^{n+1} \to \mathbf{C}_L$ is a functor. Define $|\tilde{\prod} F| \colon |\mathbf{C}|^{2n} \to |\mathbf{C}|$ to be

$$\mathrm{RK}_\pi(L \circ |F| \circ (id_{|\mathbf{C}|^{2n}} \times \Delta_{|\mathbf{C}|})),$$

$$\begin{aligned}
(\![\vec{\alpha} \vdash \alpha_i]\!)(X_1, Y_1, \ldots, X_n, Y_n) &= Y_i \\
(\![\vec{\alpha} \vdash 1]\!)(X_1, Y_1, \ldots, X_n, Y_n) &= 1 \\
(\![\vec{\alpha} \vdash \sigma \times \tau]\!)(X_1, Y_1, \ldots, X_n, Y_n) &= (\![\vec{\alpha} \vdash \sigma]\!)(X_1, Y_1, \ldots, X_n, Y_n) \otimes (\![\vec{\alpha} \vdash \tau]\!)(X_1, Y_1, \ldots, X_n, Y_n) \\
(\![\vec{\alpha} \vdash \sigma + \tau]\!)(X_1, Y_1, \ldots, X_n, Y_n) &= (\![\vec{\alpha} \vdash \sigma]\!)(X_1, Y_1, \ldots, X_n, Y_n) + (\![\vec{\alpha} \vdash \tau]\!)(X_1, Y_1, \ldots, X_n, Y_n) \\
(\![\vec{\alpha} \vdash \sigma \to \tau]\!)(X_1, Y_1, \ldots, X_n, Y_n) &= (\![\vec{\alpha} \vdash \sigma]\!)(Y_1, X_1, \ldots, Y_n, X_n) \rightharpoonup (\![\vec{\alpha} \vdash \tau]\!)(X_1, Y_1, \ldots, X_n, Y_n) \\
(\![\vec{\alpha} \vdash \textstyle\prod \beta.\, \sigma]\!) &= i \circ \tilde{\textstyle\prod}(\![\vec{\alpha}, \beta \vdash \sigma]\!) \\
(\![\vec{\alpha} \vdash \mathrm{rec}\ \beta.\, \sigma]\!) &= \mathrm{Fix}(\![\vec{\alpha}, \beta \vdash \sigma]\!)
\end{aligned}$$

Fig. 3. Auxiliary interpretation of types as functors.

where $\Delta_{|\mathbf{C}|} : |\mathbf{C}| \to |\mathbf{C}|^2$ is the diagonal, and for $\vec{f} : \vec{X}' \to \vec{X}$ and $\vec{g} : \vec{Y} \to \vec{Y}'$, define

$$\tilde{\textstyle\prod} F_1(f_1, g_1, \ldots, f_n, g_n) : |\tilde{\textstyle\prod} F|(X_1, Y_1, \ldots, X_n, Y_n) \to |\tilde{\textstyle\prod} F|(X_1', Y_1', \ldots, X_n', Y_n')$$

to be the total map corresponding to the family

$$\begin{aligned}
(|\tilde{\textstyle\prod} F|(X_1, Y_1, \ldots, X_n, Y_n) &\rightharpoonup F(X_1, Y_1, \ldots, X_n, Y_n, X, X) \\
&\rightharpoonup F(X_1', Y_1', \ldots, X_n', Y_n', X, X))_{X \in |\mathbf{C}|}
\end{aligned}$$

where the first arrow is the counit of the adjunction of Lemma 4.11 and the second is $F(f_1, g_1, \ldots, f_n, g_n, id_X, id_X)$. Then $(|\tilde{\prod} F|, \tilde{\prod} F_1)$ defines a functor

$$\tilde{\textstyle\prod} F : (\mathbf{C}_L{}^{\mathrm{op}} \times \mathbf{C}_L)^n \to \mathbf{C}.$$

Theorem 4.13. FPC can be modelled soundly in any strict FPC-model, and PolyFPC can be modelled soundly in any strict PolyFPC-model.

By soundness of the interpretation, we mean that if $t \Downarrow v$, then $[\![t]\!] = [\![v]\!]$.

In the following, we will describe the interpretation of PolyFPC into a PolyFPC model, and prove it to be sound. The interpretation of FPC into FPC models is the restriction of the interpretation of PolyFPC to FPC, which will make sense in any strict FPC model.

The interpretation is defined as follows. Open types will be interpreted as maps $|\mathbf{C}_L|^n \to |\mathbf{C}_L|$ or, equivalently, functors $|\mathbf{C}_L|^n \to \mathbf{C}_L$. However, for the interpretation of recursive types, we first define an auxiliary interpretation of open types as functors $(\![\vec{\alpha} \vdash \sigma]\!) : (\mathbf{C}_L{}^{\mathrm{op}} \times \mathbf{C}_L)^n \to \mathbf{C}_L$ as in Figure 3. In the figure, $\otimes$ refers to the tensor product induced on $\mathbf{C}_L$ by the product on $\mathbf{C}$ (on objects this is just the product) and $+$ refers to the coproduct inherited from $\mathbf{C}$. The $\tilde{\prod}$ refers to the definition in Lemma 4.12.

We can now define for any open type $\vec{\alpha} \vdash \sigma$ with n free variables the interpretation $[\![\vec{\alpha} \vdash \sigma]\!] : |\mathbf{C}_L|^n \to |\mathbf{C}_L|$ to be, as advertised,

$$[\![\vec{\alpha} \vdash \sigma]\!](\vec{X}) = (\![\vec{\alpha} \vdash \sigma]\!)(X_1, X_1, \ldots, X_n, X_n).$$

In particular, we get

$$[\![\vec{\alpha} \vdash \textstyle\prod \beta.\, \sigma]\!] = \mathrm{RK}_\pi(L \circ [\![\vec{\alpha}, \beta \vdash \sigma]\!]). \tag{2}$$

An open term $\vec{\alpha} \mid \vec{x} : \vec{\sigma} \vdash t : \tau$ will be interpreted as an indexed family of maps

$$([\![\vec{\alpha} \mid \vec{x} : \vec{\sigma} \vdash t : \tau]\!]_{\vec{X}} : \bigotimes_i [\![\vec{\alpha} \vdash \sigma_i]\!](\vec{X}) \rightharpoonup [\![\vec{\alpha} \vdash \tau]\!](\vec{X}))_{\vec{X} \in |\mathbf{C}_L|^n}$$

$$
\begin{aligned}
[\![\vec{\alpha} \mid \vec{x}\colon \vec{\sigma} \vdash x_i\colon \sigma_i]\!]_{\vec{X}} &= \pi_L^{n,i}\\
[\![\vec{\alpha} \mid \vec{x}\colon \vec{\sigma} \vdash \star\colon I]\!]_{\vec{X}} &= i(\star)\\
[\![\vec{\alpha} \mid \vec{x}\colon \vec{\sigma} \vdash \mathtt{inl}\, t\colon \tau+\tau']\!]_{\vec{X}} &= inl_{[\![\vec{\alpha}\vdash\tau]\!],[\![\vec{\alpha}\vdash\tau']\!]} \circ [\![\vec{\alpha} \mid \vec{x} \vdash t\colon \tau]\!]_{\vec{X}}\\
[\![\vec{\alpha} \mid \vec{x}\colon \vec{\sigma} \vdash \mathtt{inr}\, t\colon \tau+\tau']\!]_{\vec{X}} &= inr_{[\![\vec{\alpha}\vdash\tau]\!],[\![\vec{\alpha}\vdash\tau']\!]} \circ [\![\vec{\alpha} \mid \vec{x} \vdash t\colon \tau']\!]_{\vec{X}}\\
[\![\vec{\alpha} \mid \vec{x}\colon \vec{\sigma} \vdash \mathtt{case}\, t\, \mathtt{of}\, \mathtt{inl}\, x.t'\, \mathtt{of}\, \mathtt{inr}\, x.t''\colon \omega]\!]_{\vec{X}} &= [[\![\vec{\alpha} \mid \vec{x}\colon \vec{\sigma}, x\colon \tau \vdash t'\colon \omega]\!]_{\vec{X}},\\
&\qquad [\![\vec{\alpha} \mid \vec{x}\colon \vec{\sigma}, x\colon \tau' \vdash t''\colon \omega]\!]_{\vec{X}}]\\
&\qquad \circ d \circ \langle id_{\bigotimes_i [\![\vec{\alpha}\vdash\sigma_i]\!](\vec{X})}, [\![\vec{\alpha} \mid \vec{x}\colon \vec{\sigma} \vdash t]\!]_{\vec{X}}\rangle_L\\
[\![\vec{\alpha} \mid \vec{x}\colon \vec{\sigma} \vdash \langle t,t'\rangle\colon \tau\times\tau']\!]_{\vec{X}} &= \langle [\![\vec{\alpha} \mid \vec{x}\colon \vec{\sigma} \vdash t\colon \tau]\!]_{\vec{X}}, [\![\vec{\alpha} \mid \vec{x}\colon \vec{\sigma} \vdash t'\colon \tau']\!]_{\vec{X}}\rangle_L\\
[\![\vec{\alpha} \mid \vec{x}\colon \vec{\sigma} \vdash \pi_1(t)\colon \tau]\!]_{\vec{X}} &= \pi_L^{2,1} \circ [\![\vec{\alpha} \mid \vec{x}\colon \vec{\sigma} \vdash t\colon \tau\times\tau']\!]_{\vec{X}}\\
[\![\vec{\alpha} \mid \vec{x}\colon \vec{\sigma} \vdash \pi_2(t)\colon \tau']\!]_{\vec{X}} &= \pi_L^{2,2} \circ [\![\vec{\alpha} \mid \vec{x}\colon \vec{\sigma} \vdash t\colon \tau\times\tau']\!]_{\vec{X}}\\
[\![\vec{\alpha} \mid \vec{x}\colon \vec{\sigma} \vdash \lambda y\colon \sigma'.t\colon \sigma'\to\tau]\!]_{\vec{X}} &= i([\![\vec{\alpha} \mid \vec{x}\colon \vec{\sigma}, y\colon \sigma' \vdash t\colon \tau]\!]_{\vec{X}}{}^{\dagger})\\
[\![\vec{\alpha} \mid \vec{x}\colon \vec{\sigma} \vdash t(t')\colon \tau']\!]_{\vec{X}} &= ev_{[\![\vec{\alpha}\vdash\sigma]\!](\vec{X}),[\![\vec{\alpha}\vdash\tau]\!](\vec{X})}\circ\\
&\qquad \langle [\![\vec{\alpha} \mid \vec{x}\colon \vec{\sigma} \vdash t]\!]_{\vec{X}}, [\![\vec{\alpha} \mid \vec{x}\colon \vec{\sigma} \vdash t']\!]_{\vec{X}}\rangle_L\\
[\![\vec{\alpha} \mid \vec{x}\colon \vec{\sigma} \vdash \mathtt{intro}\, t\colon \mathrm{rec}\ \beta.\tau]\!]_{\vec{X}} &= fold \circ [\![\vec{\alpha} \mid \vec{x}\colon \vec{\sigma} \vdash t\colon \tau[\mathrm{rec}\ \beta.\tau/\beta]]\!]_{\vec{X}}\\
[\![\vec{\alpha} \mid \vec{x}\colon \vec{\sigma} \vdash \mathtt{elim}\, t\colon \tau[\mathrm{rec}\ \beta.\tau/\beta]]\!]_{\vec{X}} &= unfold \circ [\![\vec{\alpha} \mid \vec{x}\colon \vec{\sigma} \vdash t\colon \mathrm{rec}\ \beta.\tau]\!]_{\vec{X}}\\
[\![\vec{\alpha} \mid \vec{x}\colon \vec{\sigma} \vdash \Lambda\beta.t\colon \textstyle\prod\beta.\tau]\!]_{\vec{X}} &= i([\![\vec{\alpha},\beta \mid \vec{x}\colon \vec{\sigma} \vdash t\colon \tau]\!]^{\dagger}_{\vec{X}})\\
[\![\vec{\alpha} \mid \vec{x}\colon \vec{\sigma} \vdash t(\tau')\colon \tau[\tau'/\beta]]\!]_{\vec{X}} &= ev_{[\![\vec{\alpha}\vdash\tau']\!](\vec{X})} \circ [\![\vec{\alpha} \mid \vec{x}\colon \vec{\sigma} \vdash t\colon \textstyle\prod\beta.\tau]\!]_{\vec{X}}.
\end{aligned}
$$

Fig. 4. Interpretation of PolyFPC terms.

in $\mathbf{C}_L$. The interpretation is defined by induction over the structure of the term, and is presented in Figure 4, where typings as in Figure 1 are assumed. We will now explain the notation used in Figure 4.

Notice first that since the $\otimes$ is given on objects by the product from $\mathbf{C}$ and I: the neutral object for $\otimes$ is given by the terminal object in $\mathbf{C}$; we have projections out of tensor products $\pi_L^{n,i} = i(\pi^{n,i})\colon \bigotimes_{j\leqslant n} X_j \rightharpoonup X_i$ (for $\pi^{n,i}$ the i'th projection in $\mathbf{C}$); we have for any object X a map $i(\star)\colon X \rightharpoonup I$ given by $\star$ the unique map in $\mathbf{C}$ to the terminal object; and we have a diagonal $i(\Delta_X)\colon X \to X\otimes X$ in $\mathbf{C}_L$, where Δ_X is the diagonal in $\mathbf{C}$. The latter can be used to define a pairing $\langle f,g\rangle_L\colon X \rightharpoonup Y\otimes Z$ of maps in $\mathbf{C}_L$ as $f\otimes g\circ i(\Delta_X)$, but we need to bear in mind that all this does *not* in general define a product structure on $\mathbf{C}_L$.

In the interpretation of the introduction rules for coproducts, *inl* and *inr* refer to the coprojections. In the interpretation of case, the notation d refers to the isomorphism

$$\textstyle\bigotimes_i [\![\vec{\alpha}\vdash\sigma_i]\!](\vec{X}) \otimes [\![\tau+\tau']\!](\vec{X}) \xrightarrow{\cong} \bigotimes_i [\![\vec{\alpha}\vdash\sigma_i]\!](\vec{X}) \otimes [\![\tau]\!](\vec{X}) + \bigotimes_i [\![\vec{\alpha}\vdash\sigma_i]\!](\vec{X}) \otimes [\![\tau']\!](\vec{X})$$

(Lemma 4.10) and the notation $[-,-]$ refers to copairing.

Given

$$([\![\vec{\alpha} \mid \vec{x}:\vec{\sigma}, y:\sigma' \vdash t:\tau]\!]_{\vec{X}} : \bigotimes_i [\![\vec{\alpha}\vdash\sigma_i]\!](\vec{X}) \otimes [\![\vec{\alpha}\vdash\sigma']\!](\vec{X}) \rightharpoonup [\![\vec{\alpha}\vdash\tau]\!](\vec{X}))_{\vec{X}\in|\mathbf{C}_L|^n},$$

the interpretation of $\lambda y:\sigma'.t$ is defined using the family of corresponding total maps

$$([\![\vec{\alpha} \mid \vec{x}:\vec{\sigma}, y:\sigma' \vdash t:\tau]\!]_{\vec{X}}{}^{\dagger} : \bigotimes_i [\![\vec{\alpha}\vdash\sigma_i]\!](\vec{X}) \to \Big[[\![\vec{\alpha}\vdash\sigma']\!](\vec{X}) \rightharpoonup [\![\vec{\alpha}\vdash\tau]\!](\vec{X})\Big])_{\vec{X}\in|\mathbf{C}_L|^n}.$$

The interpretation of function application refers to the evaluation map

$$ev_{X,Y} : [X \rightharpoonup Y] \otimes X \rightharpoonup Y,$$

which is the counit of the adjunction $X \otimes (-) \dashv X \rightharpoonup (-)$. In the interpretation of the introduction and elimination rules for recursive types, the terms *fold* and *unfold* refer to the components of the isomorphisms for recursive domain equations as in Definition 4.1. This typechecks by Lemma 4.14 below.

For the interpretation of polymorphic lambda abstraction, we suppose

$$(\llbracket \vec{\alpha}, \beta \mid \vec{x} : \vec{\sigma} \vdash t : \tau \rrbracket_{\vec{X},Y} : \bigotimes_i \llbracket \vec{\alpha}, \beta \vdash \sigma_i \rrbracket(\vec{X}, Y) \rightharpoonup \llbracket \vec{\alpha}, \beta \vdash \tau \rrbracket(\vec{X}, Y))_{(\vec{X},Y) \in |\mathbf{C}|^{n+1}}.$$

In the typing rule for $\Lambda\alpha.\, t$, we assume that β does not occur free in the σ_i's, so, by Lemma 4.14 below, $\bigotimes_i \llbracket \vec{\alpha}, \beta \vdash \sigma_i \rrbracket(\vec{X}, Y) = \bigotimes_i \llbracket \vec{\alpha} \vdash \sigma_i \rrbracket(\vec{X})$. By (2) and Lemma 4.11, the family $\llbracket \vec{\alpha}, \beta \mid \vec{x} : \vec{\sigma} \vdash t : \tau \rrbracket_{\vec{X},Y}$ corresponds to a family of total maps

$$(\llbracket \vec{\alpha}, \beta \mid \vec{x} : \vec{\sigma} \vdash t : \tau \rrbracket^{\dagger}_{\vec{X}} : \bigotimes_i \llbracket \vec{\alpha} \vdash \sigma_i \rrbracket(\vec{X}) \to \llbracket \vec{\alpha} \vdash \prod \beta .\, \tau \rrbracket(\vec{X}))_{(\vec{X}) \in |\mathbf{C}|^{n}},$$

and we define $\llbracket \vec{\alpha} \vdash \mid \vec{x} : \vec{\sigma} \vdash \Lambda\alpha.\, t : \tau \rrbracket_{\vec{X}} = \llbracket \vec{\alpha}, \beta \mid \vec{x} : \vec{\sigma} \vdash t : \tau \rrbracket^{\dagger}_{\vec{X}}$. In the type application of polymorphic terms, $t(\tau)$ is interpreted as composition of the interpretation of t and the component

$$ev_{\llbracket \vec{\alpha} \vdash \tau' \rrbracket(\vec{X})} : \llbracket \vec{\alpha} \vdash \prod \beta .\, \sigma \rrbracket(\vec{X}) \rightharpoonup \llbracket \vec{\alpha}, \beta \vdash \sigma \rrbracket(\vec{X}, \llbracket \vec{\alpha} \vdash \tau \rrbracket(\vec{X}))$$

of the counit of the adjunction of Lemma 4.11. This type checks by Lemma 4.14 below.

Lemma 4.14. If $\vec{\alpha} \vdash \sigma$, then $\llbracket \vec{\alpha}, \beta \vdash \sigma \rrbracket = \llbracket \vec{\alpha} \vdash \sigma \rrbracket \circ \pi$, where π is the projection $\pi : |\mathbf{C}|^{n+1} \to |\mathbf{C}|^{n}$. For $\vec{\alpha}, \beta \vdash \sigma$ and $\vec{\alpha} \vdash \tau$ types of PolyFPC,

$$\llbracket \vec{\alpha} \vdash \sigma[\tau/\beta] \rrbracket = \llbracket \vec{\alpha}, \beta \vdash \sigma \rrbracket \circ \langle id_{|\mathbf{C}|^n}, \llbracket \vec{\alpha} \vdash \tau \rrbracket \rangle,$$

and if $\vec{\alpha}, \beta \mid \vec{x} : \vec{\sigma} \vdash t : \omega$, then

$$\llbracket \vec{\alpha} \mid \vec{x} : \vec{\sigma}[\tau/\beta] \vdash t[\tau/\beta] : \omega[\tau/\beta] \rrbracket_{\vec{X}} = \llbracket \vec{\alpha}, \beta \mid \vec{x} : \vec{\sigma} \vdash t : \omega \rrbracket_{(\vec{X}, \llbracket \vec{\alpha} \vdash \tau \rrbracket(\vec{X}))}.$$

Proof. This is an easy induction on τ and t, respectively. The case of recursive types in the first induction follows up to isomorphism from Lemma 4.2, which we have assumed to be an identity. The case of polymorphic types follows from the Beck–Chevalley condition in the usual way, and, again to get equality in the lemma, our assumption that the isomorphism of the Beck–Chevalley condition is an identity. □

By construction, terms are, in general, interpreted in $\mathbf{C}_L$, but values are interpreted in $\mathbf{C}$.

Lemma 4.15. For any value v of any closed type σ of PolyFPC, the interpretation $\llbracket v \rrbracket$ considered as a map from 1 to $\llbracket \sigma \rrbracket$ is total.

Proof. The proof is an easy induction on v. □

Lemma 4.16. Suppose $\vec{\alpha} \mid \vec{x} : \vec{\sigma}, x : \sigma' \vdash t : \tau$ is an PolyFPC term and $v : \sigma'$ is a value. Then

$$\llbracket \vec{\alpha} \mid \vec{x} : \vec{\sigma} \vdash t[v/x] \rrbracket = \llbracket \vec{\alpha} \mid \vec{x} : \vec{\sigma}, x : \sigma' \vdash t \rrbracket \circ \langle id_{\otimes_i \llbracket \sigma_i \rrbracket}, \llbracket \vec{\alpha} \mid \vec{x} : \vec{\sigma} \vdash v \rrbracket \rangle_L.$$

Proof. The proof is again an easy structural induction on t, but we will give the case of lambda abstractions since it shows why the reindexing only holds for values.

Suppose we are given $\vec{\alpha} \mid \vec{x} : \vec{\sigma}, x : \sigma', y : \sigma'' \vdash t : \tau$, and $v : \sigma'$ is a value. By the induction hypothesis,

$$\llbracket \vec{\alpha} \mid \vec{x} : \vec{\sigma}, y : \sigma'' \vdash t[v/x] : \tau \rrbracket = \llbracket \vec{\alpha} \mid \vec{x} : \vec{\sigma}, x : \sigma', y : \sigma'' \vdash t : \tau \rrbracket \circ (\langle id_{\otimes_i \llbracket \vec{\alpha} \vdash \sigma_i \rrbracket}, \llbracket v \rrbracket \rangle_L \times id_{\llbracket \vec{\alpha} \vdash \sigma'' \rrbracket}).$$

The map corresponding to the left-hand side under the adjunction for Kleisli function space is the interpretation of $\lambda y : \sigma''. t[v/x]$. Using Lemma 4.9, the right-hand side corresponds to

$$\llbracket \vec{\alpha} \mid \vec{x} : \vec{\sigma}, x : \sigma' \vdash \lambda y : \sigma''. t : \tau \rrbracket \circ \langle id_{\otimes_i \llbracket \vec{\alpha} \vdash \sigma_i \rrbracket}, \llbracket v \rrbracket \rangle_L$$

since, by Lemma 4.15, $\langle id_{\otimes_i \llbracket \vec{\alpha} \vdash \sigma_i \rrbracket}, \llbracket v \rrbracket \rangle_L$ is a total map. □

Finally, we will sketch the proof of Theorem 4.13.

Proof of Theorem 4.13. We need to show that if $t \Downarrow v$, then $\llbracket t \rrbracket = \llbracket v \rrbracket$. The proof is by induction on the derivation of $t \Downarrow v$. Most cases are easy, so we will just go through a few of them.

Consider the rule

$$\frac{e \Downarrow \langle v, v' \rangle}{\pi_1(e) \Downarrow v}.$$

By the induction hypothesis, $\llbracket e \rrbracket = \langle \llbracket v \rrbracket, \llbracket v' \rrbracket \rangle_L$, and by Lemma 4.15, $\llbracket v \rrbracket$ and $\llbracket v' \rrbracket$ are total maps. Since pairing and projections are inverses for total maps, $\llbracket \pi_1(e) \rrbracket = \llbracket v \rrbracket$.

For the rule

$$\frac{e \Downarrow \lambda x. e'' \qquad e' \Downarrow v \qquad e''[v/x] \Downarrow v'}{e(e') \Downarrow v'},$$

we compute

$$\begin{aligned} \llbracket e(e') \rrbracket &= ev \circ \langle \llbracket \lambda x : e'' \rrbracket, \llbracket v \rrbracket \rangle_L \\ &= \llbracket e'' \rrbracket \circ \llbracket v \rrbracket \\ &= \llbracket e''[v/x] \rrbracket. \end{aligned}$$

The first equation follows from the induction hypothesis; the second from general arguments about adjunctions; and the last from Lemma 4.16.

To prove soundness of the rule

$$\frac{t \Downarrow \Lambda\alpha. t' \qquad t'[\tau/\alpha] \Downarrow v}{t(\tau) \Downarrow v},$$

we observe that $\llbracket (\Lambda\alpha. t')(\tau) \rrbracket = \llbracket \alpha \mid - \vdash t' \rrbracket_{\llbracket \tau \rrbracket}$, which by Lemma 4.14 is $\llbracket t'[\tau/\alpha] \rrbracket$. □

5. Modelling FPC in categories of coalgebras

In this section we address the problem of interpreting the intuitionistic calculus FPC into parametric models of the linear calculus PILL$_Y$. Before presenting our solution, we will discuss the domain theoretic intuition.

As mentioned in Section 2, the domain theoretic intuition for the PILL$_Y$ is that the types are pointed cpos, and, indeed, the notion of a PILL$_Y$ model axiomatises properties of the category $\mathbf{Cppo}_\perp$ of pointed cpos with strict continuous maps and the lifting comonad on $\mathbf{Cppo}_\perp$.

On the other hand, the notion of an FPC model axiomatises properties of the category **Cpo** of complete partial orders that do not necessarily have a least element, and continuous maps between them. The problem in constructing FPC models from parametric PILL$_Y$ models is that in PILL$_Y$ there is no way of talking of cpos that are not pointed.

Thinking of domain theory, this may not seem like such a big problem. After all, one can go from pointed cpos to cpos *via* an unlifting operation inverse to the lifting operation. But such unlifting operations do not exist for general PILL$_Y$ models. Consider, for example, the admissible per given by the equivalence classes $\{\{\perp, b\}, \{a\}\}$ for some elements a, b such that $\perp < a < b$. This per is not isomorphic to the lift of any other per.

Our solution is to construct the category corresponding to **Cpo** from a PILL$_Y$ model as the co-Eilenberg–Moore category for the lifting comonad. Recall that the co-Eilenberg–Moore category for a comonad ! on a category $\mathbf{C}$ is the category whose objects are coalgebras for the comonad (maps $\xi : X \to !X$ satisfying $\epsilon \circ \xi = id$ and $(!\xi) \circ \xi = \delta \circ \xi$, for ϵ, δ are counit and comultiplication) and whose morphisms are the $\mathbf{C}$ morphisms that commute with the coalgebra structure. If $\xi : X \to !X$ is a coalgebra, X is said to be the *carrier* of the monad. We write $\mathbf{C}^!$ for the co-Eilenberg–Moore category for ! on $\mathbf{C}$. The domain theoretic intuition for why constructing the category of predomains as the co-Eilenberg–Moore category is right is that $(\mathbf{Cppo}_\perp)^!$ is isomorphic to **Cpo**, and later we shall prove a similar theorem in the theory of admissible pers (Proposition 7.2 and Theorem 7.4).

We now aim to prove that if $(\mathbf{C}, !)$ is a parametric PILL$_Y$-model, then $(\mathbf{C}^!, L)$ is an FPC model. First we need the following lemma.

Lemma 5.1. Let ! be a comonad on a category $\mathbf{C}$. Using L to denote the monad induced by ! on the co-Eilenberg–Moore category $\mathbf{C}^!$, the Kleisli category $(\mathbf{C}^!)_L$ for L is isomorphic to the category that has the same objects as $\mathbf{C}^!$ but has as morphisms from $\xi : X \to !X$ to $\chi : Y \to !Y$ all morphisms of $\mathbf{C}$ from X to Y, and composition and identity as in $\mathbf{C}$.

Proof. By definition, $(\mathbf{C}^!)_L$ has the same objects as $\mathbf{C}^!$ and has as morphisms from $\xi : X \to !X$ to $\chi : Y \to !Y$ coalgebra morphisms from ξ to $\delta_Y : !Y \to !!Y$. Since δ is the cofree coalgebra on Y, such maps correspond to morphisms from X to Y in $\mathbf{C}$, as required. It is a simple exercise to show that this correspondence preserves composition and identity. □

Lemma 5.2. Suppose $(\mathbf{C}, !)$ is a linear category. Then the induced monad L on $\mathbf{C}^!$ is strong and commutative, and the category $\mathbf{C}^!$ is cartesian and has finite coproducts. The category $(\mathbf{C}^!)_L$ has Kleisli exponentials in the sense of Definition 4.7.

Proof. In this proof, since $\mathbf{C}$ is a linear category, we write $f : X \multimap Y$ for morphisms in $\mathbf{C}$ and reserve $X \to Y$ as a shorthand for $!X \multimap Y$.

In the definition of the notion of linear category, ! is a symmetric monoidal comonad on $\mathbf{C}$, so L becomes a symmetric monoidal monad on the symmetric monoidal category $\mathbf{C}^!$. A theorem by Kock (Kock 1972) now shows that it is commutative.

As the product and coproduct structure is well known (the proof can be found in, for example, Benton (1995, Lemma 9)), we will just show the constructions. Given $\xi : X \multimap !X$ and $\xi' : X' \multimap !X'$, the product of ξ and ξ' is

$$X \otimes Y \xrightarrow{\xi \otimes \xi'} !X \otimes !Y \xrightarrow{m} !(X \otimes Y)$$

where m is the comparison map, which is part of the structure of a symmetric monoidal comonad. The terminal object is the coalgebra $m_I : I \multimap !I$, which is also part of the structure of a symmetric monoidal comonad.

The coproduct is

$$X + Y \xrightarrow{\xi + \xi'} !X + !Y \xrightarrow{[!\mathtt{inl}, !\mathtt{inr}]} !(X + Y)\,.$$

The functorial actions of the product and coproduct are simply given by the tensor and coproduct from $\mathbf{C}$, respectively.

For $\xi : X \multimap !X$ a coalgebra, the functor $\xi \rightharpoonup (-)$ maps $\chi : Y \multimap !Y$ to the cofree coalgebra $\delta : !(X \multimap Y) \multimap !!(X \multimap Y)$. This satisfies the required universal property as can be seen from the following isomorphisms of homsets

$$\mathbf{C}^!(\xi', \xi \rightharpoonup \chi) \cong \mathbf{C}(X', X \multimap Y) \cong \mathbf{C}(X' \otimes X, Y) \cong (\mathbf{C}^!)_L(\xi' \times \xi, \chi),$$

which hold for any coalgebra $\xi' : X' \multimap !X'$. □

Lemma 5.2 is the first step towards showing that $(\mathbf{C}^!, L)$ is an FPC model when $(\mathbf{C}, !)$ is a PILL$_Y$ model. The next step is to identify a collection of recursive domain equations on $(\mathbf{C}^!)_L$ that can be solved.

Definition 5.3. A functor $F_{\mathrm{coalg}} : ((\mathbf{C}^!)^{\mathrm{op}} \times \mathbf{C}^!)^n \to \mathbf{C}^!$ is a lift of $F : ((\mathbf{C})^{\mathrm{op}} \times \mathbf{C})^n \to \mathbf{C}$ to coalgebras if the diagram

$$\begin{tikzcd}
((\mathbf{C}^!)^{\mathrm{op}} \times \mathbf{C}^!)^n \arrow[r, "F_{\mathrm{coalg}}"] \arrow[d] & \mathbf{C}^! \arrow[d] \\
(\mathbf{C}^{\mathrm{op}} \times \mathbf{C})^n \arrow[r, "F"] & \mathbf{C}
\end{tikzcd}$$

commutes, where the vertical functors are the obvious forgetful functors. Similarly, we say that $F_{\mathrm{coalg}} : \mathbf{C}^! \to \mathbf{C}^!$ is a lift of $F : \mathbf{C} \to \mathbf{C}$ to coalgebras if $U \circ F_{\mathrm{coalg}} = F \circ U$ for U the forgetful functor.

Note that F_{coalg} is a lift of F, so F determines F_{coalg} on carriers in the sense that if ξ, ξ' are coalgebras with the same carrier, then $F_{\mathrm{coalg}}(\xi)$ and $F_{\mathrm{coalg}}(\xi')$ also have the same carrier. But F can also be seen as an extension of F_{coalg} to maps that are not necessarily maps of coalgebras. The latter means that if F_{coalg} is a lift of F, then F_{coalg} extends to a functor $(F_{\mathrm{coalg}}, F)\colon (((\mathbf{C}^!)_L)^{\mathrm{op}} \times (\mathbf{C}^!)_L)^n \to (\mathbf{C}^!)_L$, whose action on morphisms is given by F (by Lemma 5.1).

We show that for a parametric PILL$_Y$ model $(\mathbf{C}, !)$, the collection of recursive domain equations on $(\mathbf{C}^!)_L$ given by pairs (F_{coalg}, F) where F_{coalg} lifts F to coalgebras and F is a strong functor as in Section 4.1 can be solved. For the proof, we will apply Theorem 4.5, so we begin by considering initial algebras.

Proposition 5.4. Suppose $\mathbf{C}$ is a category and ! is a comonad on $\mathbf{C}$, and suppose that $F_{\mathrm{coalg}} : \mathbf{C}^! \to \mathbf{C}^!$ is a lift of $F : \mathbf{C} \to \mathbf{C}$ to coalgebras. If F has an initial algebra, then so does F_{coalg}. Moreover, the image of the initial algebra for F_{coalg} under the forgetful functor $U : \mathbf{C}^! \to \mathbf{C}$ is the initial algebra for F.

Proof. In this proof, whenever we write $F_{\mathrm{coalg}}(\xi)$, we mean the functor F_{coalg} applied to a coalgebra considered as an object of $\mathbf{C}^!$. The action of F_{coalg} on morphisms of coalgebras is always denoted F, as F and F_{coalg} agree on morphisms. The maps ϵ, δ denote counit and comultiplication for the comonad !.

Suppose $in\colon F(\mu X. F(X)) \to \mu X. F(X)$ is an initial algebra for F. We will show that $\xi : \mu X. F(X) \to !\mu X. F(X)$ defined as the unique map making the diagram

$$\begin{tikzcd}
F(\mu X. F(X)) \arrow[rrr, "in"] \arrow[d, "F(\xi)"'] & & & \mu X. F(X) \arrow[d, "\xi"] \\
F(!\mu X. F(X)) \arrow[r, "F_{\mathrm{coalg}}(\delta)"] & !F(!\mu X. F(X)) \arrow[r, "!F(\epsilon)"] & !F(\mu X. F(X)) \arrow[r, "!in"] & !\mu X. F(X)
\end{tikzcd}$$

commute is the carrier of an initial algebra for F_{coalg}. First we must show that ξ in fact defines an object of $\mathbf{C}^!$, that is, that $\epsilon \circ \xi$ is the identity and that $\delta \circ \xi = !\xi \circ \xi$. The algebra is given by in, and we must show that this actually defines a map in $\mathbf{C}^!$, that is, that the diagram

$$\begin{tikzcd}
F(\mu X. F(X)) \arrow[r, "in"] \arrow[d, "F_{\mathrm{coalg}}(\xi)"'] & \mu X. F(X) \arrow[d, "\xi"] \\
!(F(\mu X. F(X))) \arrow[r, "!in"] & !\mu X. F(X)
\end{tikzcd} \tag{3}$$

commutes. Finally, to show that this is an initial algebra, we should show that given any other algebra, that is, a commutative diagram of the form

$$\begin{tikzcd}
F(X) \arrow[r, "f"] \arrow[d, "F_{\mathrm{coalg}}(\chi)"'] & X \arrow[d, "\chi"] \\
!F(X) \arrow[r, "!f"] & !X
\end{tikzcd} \tag{4}$$

there exists a unique map $g : \mu X.F(X) \to X$ making the diagram

$$\begin{tikzcd}
F(\mu X.F(X)) \arrow[rr, "in"] \arrow[dr, "F(g)"] \arrow[dd, "F_{\mathrm{coalg}}(\xi)"'] & & \mu X.F(X) \arrow[dr, "g"] \arrow[dd, "\xi"] & \\
& F(X) \arrow[rr, "f"] \arrow[dd, "F_{\mathrm{coalg}}(\chi)"] & & X \arrow[dd, "\chi"] \\
!(F(\mu X.F(X))) \arrow[rr, "!in"] \arrow[dr, "!F(g)"'] & & !\mu X.F(X) \arrow[dr, "!g"] & \\
& !F(X) \arrow[rr, "!f"] & & !X
\end{tikzcd} \tag{5}$$

commute. Of course, since *in* is an initial algebra for F, there exists a unique map g making the top square (5) commute, and we just need to show that the rest of the diagram commutes with g being this map. Showing all this is an easy diagram chase, which can be found in Appendix A. □

Theorem 5.5. Suppose **C** is a category and ! is a comonad on **C**, and let L denote the monad on $\mathbf{C}^!$ induced by !. Suppose **C** has solutions to recursive domain equations as given by the collection of functors $\mathscr{F}$, and suppose $F : (\mathbf{C}^{\mathrm{op}} \times \mathbf{C})^{n+1} \to \mathbf{C}$ is in $\mathscr{F}$. If F has a lift to coalgebras F_{coalg}, then Fix$F$ also has a lift Fix(F_{coalg}). Moreover, for any $2n$ vector $\vec{\xi}$ of coalgebras with carriers denoted $\vec{X}$, the isomorphism

$$F \circ \langle id, \mathrm{Fi\check{x}}F\rangle(\vec{X}) \cong \mathrm{Fix}F(\vec{X})$$

is an isomorphism of coalgebras

$$F_{\mathrm{coalg}} \circ \langle id, \mathrm{Fix}(\check{F}_{\mathrm{coalg}})\rangle(\vec{\xi}) \cong \mathrm{Fix}(F_{\mathrm{coalg}})(\vec{\xi}).$$

Proof. The first half of the theorem states that the class of domain equations $\mathscr{F}' \subseteq \mathscr{F}$ consisting of functors with lifts to coalgebras has solutions, and we will use Theorem 4.5 to prove this. By assumption, all functors $F : \mathbf{C} \to \mathbf{C}$ in $\mathscr{F}$ have initial algebras whose inverses are final coalgebras, so the same condition holds for all F in $\mathscr{F}'$. We still need to verify that if $F : (\mathbf{C}^{\mathrm{op}} \times \mathbf{C})^n \times \mathbf{C} \to \mathbf{C}$ is in $\mathscr{F}'$, so is $\mu X.F(-, X) : (\mathbf{C}^{\mathrm{op}} \times \mathbf{C})^n \to \mathbf{C}$ formed using Lemma 4.4. So suppose $F_{\mathrm{coalg}} : (\mathbf{C}^{!\mathrm{op}} \times \mathbf{C}^!)^n \times \mathbf{C}^! \to \mathbf{C}^!$ is a lift of F to coalgebras. By Proposition 5.4, each $F_{\mathrm{coalg}}(\vec{\xi}, \vec{\xi}', -) : \mathbf{C}^! \to \mathbf{C}^!$ has an initial algebra, so we can form $\mu\xi.F_{\mathrm{coalg}}(-, \xi) : (\mathbf{C}^{!\mathrm{op}} \times \mathbf{C}^!)^n \to \mathbf{C}^!$ using Lemma 4.4. We claim that

$$\begin{tikzcd}
((\mathbf{C}^!)^{\mathrm{op}} \times \mathbf{C}^!)^n \arrow[r, "\mu\xi.F_{\mathrm{coalg}}(-,\xi)"] \arrow[d] & \mathbf{C}^! \arrow[d] \\
(\mathbf{C}^{\mathrm{op}} \times \mathbf{C})^n \arrow[r, "\mu X.F(-,X)"] & \mathbf{C}
\end{tikzcd}$$

commutes, verifying $\mu X.F(-, X) \in \mathscr{F}'$. The diagram commutes on objects by the last statement of Proposition 5.4. Inspection of the proof of Proposition 5.4 shows that the unique maps out of the initial algebras given by $\mu\xi.F_{\mathrm{coalg}}(-, \xi)$ are computed as for

$\mu X.F(-,X)$, so, since the morphism parts of $\mu\xi.F_{\mathrm{coalg}}(-,\xi)$ and $\mu X.F(-,X)$ are computed using these, the diagram commutes for morphisms as well.

To prove the last statement of the theorem, we need to have a closer look at the construction in the proof of Theorem 4.5. Since the construction is pointwise, we will just consider the case of domain equations with no variables. So suppose $F_{\mathrm{coalg}} : \mathbf{C}^{!\mathrm{op}} \times \mathbf{C}^! \to \mathbf{C}^!$ is a lift of $F : \mathbf{C}^{\mathrm{op}} \times \mathbf{C} \to \mathbf{C}$ to coalgebras. Consider the functor $G = \mu X.F(-,X) : \mathbf{C}^{\mathrm{op}} \to \mathbf{C}$. Define the object X' of $\mathbf{C}$ as $\mu Y.F(G(Y),Y)$ and define $X = G(X')$. Since these are obtained as initial algebras, we get maps $h : F(X',X) \to X$ and $h' : F(X,X') \to X'$ in $\mathbf{C}$, and we can show that $(X,X',h,(h')^{-1})$ as well as (X',X,h',h^{-1}) are initial dialgebras for F. This implies that there exists an isomorphism of dialgebras $(k,k') : (X',X,h',h^{-1}) \to (X,X',h,(h')^{-1})$. Now, the solution to the recursive domain equation given by F is X with isomorphism $h \circ F(k,id) : F(X,X) \to X$.

By Proposition 5.4, we know that X and X' have coalgebra structures for !, so $F(X,X')$ and $F(X',X)$ also get coalgebra structures by application of F_{coalg}. We also know that h and h' preserve these coalgebra structures, and what we need to prove is that $h \circ F(k,id)$ does also. In fact, to do this, it is enough to show that k preserves coalgebra structures, because in this case the map $F_{\mathrm{coalg}}(k,id) = F(k,id)$ preserves coalgebra structures.

The construction of k is as follows. Using $out : G(X) \to F(X,G(X))$ to denote the final coalgebra for $F(X,-)$, we construct $a : X' \to G(X)$ as the unique map making the diagram

$$\begin{tikzcd}
X' \arrow[r, "(h')^{-1}"] \arrow[d, "a"'] & F(X,X') \arrow[d, "F(id{,}a)"] \\
G(X) \arrow[r, "out"] & F(X,G(X))
\end{tikzcd} \tag{6}$$

commute. Now, k is defined as the unique map making the diagram

$$\begin{tikzcd}
F(X,X') \arrow[rr, "h'"] \arrow[d, "F(G(k){,}k)"'] & & X' \arrow[d, "k"] \\
F(G(X),X) \arrow[r, "F(a{,}id)"] & F(X',X) \arrow[r, "h"] & X
\end{tikzcd} \tag{7}$$

commute. The map k' is defined as $G(k) \circ a$, but one can prove that $k = k'$. Since (k,k) is an isomorphism of dialgebras, k is an isomorphism also, implying that a is an isomorphism since $k = G(k) \circ a$.

We first prove that a is a map of coalgebras. Since both vertical maps in (6) are invertible, we can use initiality of out^{-1} to construct a map $b : G(X) \to X'$ satisfying $a \circ b = id$, which implies that b is the inverse of the isomorphism a. By Proposition 5.4, since the vertical maps of (6) preserve coalgebras, so does b, and so does a. This implies that both vertical maps in (7) preserve coalgebra structure, so, again by Proposition 5.4, k preserves coalgebra structures, which concludes the proof. □

Corollary 5.6. If $(\mathbf{C}, !)$ is a parametric PILL$_Y$ model, then $(\mathbf{C}^!, L)$ is an FPC model.

Proof. Using Lemma 5.2, all that remains to be proved is that there exists a rich class of recursive domain equations on $(\mathbf{C}^!)_L$ that can be solved. To do this, we consider the collection $\mathscr{G}$ of functors induced by pairs (F, F_{coalg}) where F_{coalg} is a lift of F to coalgebras and F is strong. By Theorem 4.6 and the first half of Theorem 5.5, for any pair (F, F_{coalg}), we get a pair $(\mathrm{Fix}F, \mathrm{Fix}(F_{\mathrm{coalg}}))$ providing a fixed point of the functor (F, F_{coalg}) up to an isomorphism. Moreover, the isomorphism consists of total maps, that is, maps of coalgebras. For the universal condition, since the morphisms of $(\mathbf{C}^!)_L$ are simply those of $\mathbf{C}$, and the morphism part of (F, F_{coalg}) is given by F, the universal condition of $(\mathrm{Fix}F, \mathrm{Fix}(F_{\mathrm{coalg}}))$ reduces to that of $\mathrm{Fix}F$, which is satisfied by assumption.

Finally, we must show that $\mathscr{G}$ is closed under products, coproducts and Kleisli exponentials. This can be seen from the constructions in the proof of Lemma 5.2. For example, for the case of Kleisli exponentials, the functor obtained by Lemma 4.8 is induced by the pair (F, F_{coalg}) defined as

$$\begin{aligned} F_{\mathrm{coalg}}(\xi : X \multimap !X, \xi' : Y \multimap !Y) &= \delta_{(X \multimap Y)} \\ F(X, Y) &= !(X \multimap Y). \end{aligned} \qquad \square$$

Unfortunately, the FPC model obtained from a PILL_Y model $(\mathbf{C}, !)$ need not be strict, even if we have a strict choice of solutions to recursive domain equations on $\mathbf{C}$. The problem is that a functor on $(\mathbf{C}^!)_L$ that can be represented by a pair (F, F_{coalg}) need not have a unique such representation, so the solution to a recursive domain equation given by (F, F_{coalg}) as constructed in the proofs above can depend on the representation. In the following we present a slight variant of the interpretation of Section 4.2 that gives a sound interpretation of FPC into any PILL_Y model.

We will first give an auxiliary interpretation of each FPC type $\alpha_1, \ldots, \alpha_n \vdash \sigma$ as a pair of functors

$$\begin{aligned} (\![\vec{\alpha} \vdash \sigma]\!) &: (\mathbf{C}^{\mathrm{op}} \times \mathbf{C})^n \to \mathbf{C} \\ (\![\vec{\alpha} \vdash \sigma]\!)_{\mathrm{coalg}} &: (\mathbf{C}^{!\mathrm{op}} \times \mathbf{C}^!)^n \to \mathbf{C}^! \end{aligned}$$

with the property that $(\![\vec{\alpha} \vdash \sigma]\!)_{\mathrm{coalg}}$ lifts $(\![\vec{\alpha} \vdash \sigma]\!)$. This interpretation is given in Figure 5, where in the definition of $(\![\vec{\alpha} \vdash \sigma]\!)_{\mathrm{coalg}}$, the symbols $1, +, \times$ refer to the terminal object, coproducts and products of $\mathbf{C}^!$, respectively, the notation δ_Z is used for the cofree coalgebra on Z, and U denotes the forgetful functor $\mathbf{C}^! \to \mathbf{C}$.

Using the auxiliary interpretation of types as pairs of functors, we define

$$[\![\vec{\alpha} \vdash \sigma]\!] : |(\mathbf{C}^!)_L|^n \to |(\mathbf{C}^!)_L|$$

as $[\![\vec{\alpha} \vdash \sigma]\!](\vec{\xi}) = (\![\vec{\alpha} \vdash \sigma]\!)_{\mathrm{coalg}}(\xi_1, \xi_1, \ldots, \xi_n, \xi_n)$. The interpretation of terms is exactly as in Section 4.2.

6. Interpreting FPC into PILL_Y

In this short section we show how the theory of Section 5 gives rise to an interpretation of FPC into PILL_Y.

We use $\mathbf{PILL}_Y$ to denote the category whose objects are the closed types of PILL_Y and whose morphisms from σ to τ are equivalence classes of closed PILL_Y terms of type $\sigma \multimap \tau$ under the equivalence relation relating terms if they are provably equal in LAPL

$$\begin{aligned}
\llbracket\vec{\alpha}\vdash\alpha_i\rrbracket(X_1,Y_1,\ldots,X_n,Y_n) &= Y_i\\
\llbracket\vec{\alpha}\vdash 1\rrbracket(X_1,Y_1,\ldots,X_n,Y_n) &= I\\
\llbracket\vec{\alpha}\vdash\sigma+\tau\rrbracket(X_1,Y_1,\ldots,X_n,Y_n) &= \llbracket\vec{\alpha}\vdash\tau\rrbracket(X_1,Y_1,\ldots,X_n,Y_n)\\
&\qquad+\llbracket\vec{\alpha}\vdash\tau\rrbracket(X_1,Y_1,\ldots,X_n,Y_n)\\
\llbracket\vec{\alpha}\vdash\sigma\times\tau\rrbracket(X_1,Y_1,\ldots,X_n,y_n) &= \llbracket\vec{\alpha}\vdash\tau\rrbracket(X_1,Y_1,\ldots,X_n,Y_n)\otimes\\
&\qquad\llbracket\vec{\alpha}\vdash\tau\rrbracket(X_1,Y_1,\ldots,X_n,Y_n)\\
\llbracket\vec{\alpha}\vdash\sigma\to\tau\rrbracket(X_1,Y_1,\ldots,X_n,y_n) &= !(\llbracket\vec{\alpha}\vdash\sigma\rrbracket(Y_1,X_1,\ldots,Y_n,X_n)\multimap\\
&\qquad !\llbracket\vec{\alpha}\vdash\tau\rrbracket(X_1,Y_1,\ldots,X_n,Y_n))\\
\llbracket\vec{\alpha}\vdash\mathrm{rec}\ \beta.\,\sigma\rrbracket &= \mathrm{Fix}\llbracket\vec{\alpha},\beta\vdash\sigma\rrbracket
\end{aligned}$$

$$\begin{aligned}
\llbracket\vec{\alpha}\vdash\alpha_i\rrbracket_{\mathrm{coalg}}(\xi_1,\xi_1',\ldots,\xi_n,\xi_n') &= \xi_i'\\
\llbracket\vec{\alpha}\vdash 1\rrbracket_{\mathrm{coalg}}(\xi_1,\xi_1',\ldots,\xi_n,\xi_n') &= m_I\\
\llbracket\vec{\alpha}\vdash\sigma+\tau\rrbracket_{\mathrm{coalg}}(\xi_1,\xi_1',\ldots,\xi_n,\xi_n') &= \llbracket\vec{\alpha}\vdash\sigma\rrbracket_{\mathrm{coalg}}(\xi_1,\xi_1',\ldots,\xi_n,\xi_n')\\
&\qquad+\llbracket\vec{\alpha}\vdash\tau\rrbracket_{\mathrm{coalg}}(\xi_1,\xi_1',\ldots,\xi_n,\xi_n')\\
\llbracket\vec{\alpha}\vdash\sigma\times\tau\rrbracket_{\mathrm{coalg}}(\xi_1,\xi_1',\ldots,\xi_n,\xi_n') &= \llbracket\vec{\alpha}\vdash\sigma\rrbracket_{\mathrm{coalg}}(\xi_1,\xi_1',\ldots,\xi_n,\xi_n')\times\\
&\qquad\llbracket\vec{\alpha}\vdash\tau\rrbracket_{\mathrm{coalg}}(\xi_1,\xi_1',\ldots,\xi_n,\xi_n')\\
\llbracket\vec{\alpha}\vdash\sigma\to\tau\rrbracket_{\mathrm{coalg}}(\xi_1,\xi_1',\ldots,\xi_n,\xi_n') &=\\
&\delta_{\llbracket\vec{\alpha}\vdash\sigma\rrbracket(U(\xi_1'),U(\xi_1),\ldots,U(\xi_n'),U(\xi_n))\multimap!\llbracket\vec{\alpha}\vdash\tau\rrbracket(U(\xi_1),U(\xi_1'),\ldots,U(\xi_n),U(\xi_n'))}\\
\llbracket\vec{\alpha}\vdash\mathrm{rec}\ \beta.\,\sigma\rrbracket_{\mathrm{coalg}} &= \mathrm{Fix}\llbracket\vec{\alpha},\beta\vdash\sigma\rrbracket_{\mathrm{coalg}}
\end{aligned}$$

Fig. 5. Auxiliary interpretation of FPC types in PILL$_Y$ models

using parametricity. This category is symmetric monoidal closed and the type constructor ! induces a symmetric monoidal comonad on it such that $(\mathbf{PILL}_Y, !)$ is, in fact, a linear category. As above, we will use $\mathbf{PILL}_Y^!$ to denote the co-Eilenberg–Moore category and L to denote the induced monad on $\mathbf{PILL}_Y^!$.

Theorem 6.1. The category $(\mathbf{PILL}_Y^!, L)$ is an FPC-model.

Theorem 6.1 follows from Lemma 5.2 and Proposition 4.6 together with Theorem 5.5. The collection of domain equations is in this case given by the functors

$$F : (\mathbf{PILL}_Y^{!\ \mathrm{op}} \times \mathbf{PILL}_Y^!)^n \to \mathbf{PILL}_Y^!$$

for which there exists a PILL$_Y$ type $\alpha_1,\ldots,\alpha_n,\beta_1,\ldots,\beta_n\vdash\sigma$ in which the α's occur only negatively and the β's only positively, such that we have, when $(\xi_i : \tau_i \multimap !\tau_i)_{i\leqslant n}$ and $(\chi_i : \omega_i \multimap !\omega_i)_{i\leqslant n}$ are vectors of coalgebras, the carrier of $F(\xi_1,\chi_1,\ldots,\xi_n,\chi_n)$ is $\sigma[\vec{\tau}/\vec{\alpha},\vec{\omega}/\vec{\beta}]$, and such that the action of F on morphisms of coalgebras is given by the term M_σ as in Section 4.1.

Theorem 6.1 gives an interpretation of FPC into PILL$_Y$ interpreting a closed FPC type σ as a closed PILL$_Y$ type σ^* and an FPC program $t\colon\sigma$ as a closed PILL$_Y$ term $t^* : \sigma^*$. The interpretation is sound in the sense that $t \Downarrow v$ implies t^* is provably equal to v^* in

LAPL using parametricity. Later we shall prove that this interpretation is computationally adequate (Corollary 7.9).

A priori, the interpretation does not interpret *open* FPC types and terms as PILL_Y types and terms. Instead of this, open FPC types are interpreted as maps $|\mathbf{PILL}_Y^!|^n \to |\mathbf{PILL}_Y^!|$. But the interpretation can also be presented more syntactically by interpreting an open FPC type $\vec{\alpha} \vdash \sigma$ as a pair

$$(\vec{\alpha} \vdash \sigma^*, \sigma^*_{\mathrm{coalg}} : \prod \vec{\alpha}.\, (\alpha_1 \multimap !\alpha_1) \to \ldots \to (\alpha_n \multimap !\alpha_n) \to \sigma^* \multimap !\sigma^*)$$

where the first component is a PILL_Y type with n free variables, and $\sigma^*_{\mathrm{coalg}}$ is a closed PILL_Y-term such that the statement

$$\text{if for all } i,\ f_i : \alpha_i \multimap !\alpha_i \text{ is a coalgebra, then so is } \sigma^*_{\mathrm{coalg}}\vec{\alpha}(!\vec{f}),$$

is provable in LAPL. An open FPC term, say $\vec{\alpha} \mid x : \sigma \vdash t : \tau$, is then interpreted as a term

$$t^* : \prod \vec{\alpha}.\, (\alpha_1 \multimap !\alpha_1) \to \ldots \to (\alpha_n \multimap !\alpha_n) \to \sigma^* \multimap \tau^*.$$

7. Polymorphic FPC in the per-model

As a special case, the abstract analysis of Section 5 shows that our main example, the per-model, models FPC. This particular model is interesting for three reasons: the interpretation of FPC extends to an interpretation of full PolyFPC; the concrete model has a simple presentation; and we can show that this model of PolyFPC is computationally adequate. In this section, we first give the more concrete presentation of the model, and show that it also models full PolyFPC. In Section 7.1, we present the resulting PolyFPC model in elementary terms, and computational adequacy is proved in Section 7.2.

Recall that in order to see the per-model as an example of the notion of PILL_Y-model of Section 2.1, we must interpret this definition in the internal logic of $\mathrm{RT}(D)^G$. Since $\mathrm{RT}(D)^G$ is a topos, its internal language is an expressive intuitionistic type theory in which we can interpret most constructions known from set theory (again, see Jacobs (1999), Lambek and Scott (1986) and Johnstone (2002) for details). This means, for example, that one can form the co-Eilenberg–Moore category for an internal comonad on an internal category $\mathrm{RT}(D)^G$ as an internal category. Moreover, since the arguments of Section 5 were all constructive, we can interpret all this theory in the internal logic of the topos $\mathrm{RT}(D)^G$, and this gives us an FPC model. As was the case of the PILL_Y model, the results in the internal logic of $\mathrm{RT}(D)^G$ give rise to an interpretation of FPC in the usual (external) sense. If you are only interested in the elementary description of the model, you should now skip to Section 7.1.

To understand the model, we will first give a concrete description of the co-Eilenberg–Moore category. To be precise, since we interpret the constructions of Section 5 inside $\mathrm{RT}(D)^G$, the category we are interested in is the externalisation of a co-Eilenberg–Moore category as formed in the internal logic of $\mathrm{RT}(D)^G$. Fortunately, there is no difference between considering the externalisation of the co-Eilenberg–Moore category as formed internally, and the co-Eilenberg–Moore category as formed in the usual sense from the external data, as the next lemma states. Because of this, in the following we shall not specify which of the two is meant in statements of theorems.

Lemma 7.1. Suppose $\mathbb{E}$ is a topos, $\mathbf{C}$ is an internal category in $\mathbb{E}$ and ! is an internal comonad on $\mathbf{C}$. Then the externalisation of the co-Eilenberg–Moore category as formed internally in $\mathbb{E}$ is isomorphic to the co-Eilenberg–Moore category as formed in the usual sense from the externalisation of $\mathbf{C}$ and !.

Proof. An object of the externalisation of the co-Eilenberg–Moore category as formed internally is a global element of the object of objects of $\mathbf{C}$ and a global element of the object of morphisms of $\mathbf{C}$ satisfying equations stating that the latter is a coalgebra structure on the former. This is precisely what is needed to specify an object of the co-Eilenberg–Moore category as formed in the external sense. We leave it to the interested reader to write out the isomorphisms in detail. □

We start by considering the co-Eilenberg–Moore category for ! on $\mathbf{AP}_\perp$. The first result states that this is equivalent to $\mathbf{CCP}$, the full subcategory of $\mathbf{Per}(D)$ on chain complete pers. This is a nice result as $\mathbf{CCP}$ seems a natural choice of a category of predomains in the theory of admissible pers, and so, as we have already mentioned, this can be seen as parallel to the isomorphism $(\mathbf{Cppo}_\perp)^L \cong \mathbf{Cpo}$ from classical domain theory. Like the other categories of pers in this paper, $\mathbf{CCP}$ is the externalisation of an internal category of $\mathrm{RT}(D)$, and this equivalence of categories is also true in the stronger internal sense.

Proposition 7.2. The co-Eilenberg–Moore category $\mathbf{AP}_\perp^!$ for the lifting comonad ! on $\mathbf{AP}_\perp$ is equivalent to $\mathbf{CCP}$. Moreover, the category $\mathbf{AP}_\perp^!$ as formed in the internal logic of $\mathrm{RT}(D)$ is equivalent to the internal category $\mathbf{CCP}$ in the usual sense of internal categories of $\mathrm{RT}(D)$, that is, there exist internal functors $F : \mathbf{CCP} \to \mathbf{AP}_\perp^!$ and $G : \mathbf{AP}_\perp^! \to \mathbf{CCP}$ and invertible internal natural transformations $\alpha : G \circ F \to id_{\mathbf{CCP}}$, $\beta : F \circ G \to id_{\mathbf{AP}_\perp^!}$.

Proof. We show that $\mathbf{AP}_\perp^!$ is isomorphic to the category of admissible pers R for which the equivalence class $[\perp]$ is a downward closed subset of the domain of R – that is, if $R(x,x)$, $x \leqslant y$ and $R(y,\perp)$, then $R(x,\perp)$ – and maps that preserve and reflect $[\perp]$. This category is equivalent to $\mathbf{CCP}$, with one map of the equivalence lifting a chain complete per, and the other discarding the equivalence class $[\perp]$ from an admissible per.

Recall from Section 2.2 that the comonad ! on $\mathbf{AP}_\perp$ maps an admissible per R to $\{(\perp,\perp)\} \cup \{(\langle\iota,x\rangle,\langle\iota,y\rangle) \mid R(x,y)\}$ where $\iota = \Psi(id_D)$. Suppose ξ is a coalgebra on an admissible per R, e tracks ξ and $R(x,x)$, $x \leqslant y$ and $R(y,\perp)$. Since $[\perp]_{!R} = \{\perp\}$ and since ξ is a strict map, we have $e \cdot y = \perp$, so, by monotonicity, we have $e \cdot x = \perp$. Since $\epsilon \circ \xi$ is the identity, where ϵ is the counit, $R(x,\perp)$. On the other hand, if $[\perp]$ is a downward closed subset of the domain of R, it is easy to check that the tracker

$$e(x) = \begin{cases} \perp & \text{if } y \geqslant x, R(y,\perp) \text{ for some } y \\ \langle\iota,x\rangle & \text{else} \end{cases}$$

defines a unique coalgebra structure on R. Continuity of e follows from admissibility of R.

Suppose $f : R \multimap S$ is a map between such pers with coalgebra structures, denoted ξ_R and ξ_S, respectively, and that f preserves coalgebra structure. Since f is strict, it must preserve the equivalence class of $\perp$. To see that it also reflects it, suppose $f([x]_R) = [\perp]_S$. Then also $!(f)(\xi_R([x])) = [\perp]_{!S}$, implying that $\xi_R([x]) = [\perp]_{!R}$. Clearly, this then gives us $R(x, \perp)$.

We omit the straightforward verification of the second statement of the theorem. $\square$

Note that the composition $\mathbf{CCP} \to \mathbf{AP}_\perp^! \to \mathbf{AP}_\perp$ of the equivalence and the forgetful functor is the lifting functor, as can be seen by inspection of the above proof.

Proposition 7.2 also holds on the level of relations, as we will now show. As admissible relations on chain complete pers R and S, we shall consider subsets $A \subseteq D/R \times D/S$ such that if $(x_n)_{n\in\mathbb{N}}$ and $(y_n)_{n\in\mathbb{N}}$ are increasing chains such that for all n, $R(x_n, x_n)$ and $S(y_n, y_n)$ and $([x_n]_R, [y_n]_S) \in A$, then $([\bigsqcup_n x_n]_R, [\bigsqcup_n y_n]_S) \in A$ also. Admissible relations on chain complete pers form a category $\mathbf{AdmRel}_{\mathbf{CCP}}$ where maps are pairs of maps mapping related elements to related elements as in Section 2.2. We write $A : \mathrm{AdmRel}_{\mathbf{CCP}}(R, S)$ to indicate that A is an admissible relation between chain complete pers R, S. As with $\mathbf{AdmRel}_{\mathbf{AP}_\perp}$, the category $\mathbf{AdmRel}_{\mathbf{CCP}}$ is the externalisation of an internal category of $\mathrm{RT}(D)$.

Proposition 7.3. The co-Eilenberg–Moore category for ! on $\mathbf{AdmRel}_{\mathbf{AP}_\perp}$ is equivalent to $\mathbf{AdmRel}_{\mathbf{CCP}}$. The equivalence is an externalisation of an internal equivalence of categories in $\mathrm{RT}(D)$.

Proof. We will again just show the first part of the proposition. Suppose R, S are admissible pers and (ξ, ξ') defines the coalgebra structure on $A : \mathrm{AdmRel}_{\mathbf{AP}_\perp}(R, S)$. Then ξ, ξ' define the coalgebra structures on R, S, respectively, so the analysis from the proof of Proposition 7.2 applies. We show that A is a lifted relation in the sense that $A([\perp], [\perp])$ and if $A([x], [\perp])$ then $[x] = [\perp]$, and, similarly, $A([\perp], [x])$ implies $[x] = [\perp]$. The first follows from admissibility. For the second, we suppose $A([\perp]_R, [x]_S)$. Then $!A([\perp]_{!R}, \xi'([x]_S))$, implying $\xi'([x]) = [\perp]_{!S}$, and thus $[x]_S = [\perp]_S$. So we can extend the equivalence of categories defined above to map $A : \mathbf{AdmRel}_{\mathbf{CCP}}$ to $!A$, and map $A \in \mathbf{AdmRel}^!_{\mathbf{AP}_\perp}$ to $A \setminus \{([\perp], [\perp])\}$. $\square$

Now recall that we wanted to apply the theory of Section 5 to the diagram

$$\begin{tikzcd}\mathbf{AdmRel}_{\mathbf{AP}_\perp} \arrow[r, shift left=2] \arrow[r, shift right=2] & \mathbf{AP}_\perp \arrow[l]\end{tikzcd} \tag{8}$$

considered as an internal category in the category $\mathrm{RT}(D)^G$ of reflexive graphs over the realisability topos over D. So we were interested in a concrete description of the co-Eilenberg–Moore category for the lifting comonad on (8). Fortunately, co-Eilenberg–Moore categories in $\mathrm{RT}(D)^G$ are constructed pointwise[†], so Propositions 7.2 and 7.3 imply the following theorem.

Theorem 7.4. The co-Eilenberg–Moore category for the lifting comonad on

$$\begin{tikzcd}\mathbf{AdmRel}_{\mathbf{AP}_\perp} \arrow[r, shift left=2] \arrow[r, shift right=2] & \mathbf{AP}_\perp \arrow[l]\end{tikzcd}$$

[†] Technically, constructing the co-Eilenberg–Moore category in the internal logic amounts to a limit construction, and limits in presheaf categories such as $\mathrm{RT}(D)^G$ are computed pointwise. Similar arguments are used in Robinson and Rosolini (1994).

in the category $\mathrm{RT}(D)^G$ is equivalent to

$$\begin{tikzcd}\mathbf{AdmRel}_{\mathbf{CCP}} \arrow[r, shift left=2] \arrow[r, shift right=2] & \mathbf{CCP} \arrow[l] \end{tikzcd}. \tag{9}$$

The maps of diagram (9) are the usual domain, codomain and identity relation maps, which one can show define internal functors.

Theorem 7.5. Diagram (9) considered as an internal category in $\mathrm{RT}(D)^G$ models polymorphism. As a consequence, the per model of FPC models full PolyFPC.

Proof. In this proof we shall use the notation $\mathbf{C}$ for the internal category in $\mathrm{RT}(D)^G$ given by the diagram (8), and $\mathbf{D}$ for the one given by (9). A map from $|\mathbf{D}|^n$ to $|\mathbf{D}|$ inside $\mathrm{RT}(D)^G$ is a pair of maps (f^p, f^r) where f^p is a map $|\mathbf{CCP}|^n \to |\mathbf{CCP}|$ and f^r is a map taking an n-vector of admissible relations $(A_i : \mathbf{AdmRel}(R_i, S_i))$ (admissible in the sense of objects of $\mathbf{AdmRel}_{\mathbf{CCP}}$) on objects of $\mathbf{CCP}$ and produces an admissible relation

$$f^r(\vec{A}) : \mathbf{AdmRel}(f^p(\vec{R}), f^p(\vec{S}))$$

satisfying $f^r(eq_{\vec{R}}) = eq_{f^p(\vec{R})}$, and a natural transformation from (f^p, f^r) to (g^p, g^r) is a family of maps $(t_{\vec{R}} : f^p(\vec{R}) \to g^p(\vec{R}))_{\vec{R}\in|\mathbf{CCP}|^n}$ with a common tracker, which respects relations in the sense that for all vectors $\vec{R}$, $\vec{S}$, $(A_i : \mathbf{AdmRel}(R_i, S_i))_i$, if $([x]_{f^p(\vec{R})}, [y]_{f^p(\vec{S})}) \in f^r(\vec{A})$ then $(t_{\vec{R}}([x]_{f^p(\vec{R})}), t_{\vec{S}}([y]_{f^p(\vec{S})})) \in g^r(\vec{A})$.

We show that any such functor $(f^p, f^r) : |\mathbf{D}|^{n+1} \to |\mathbf{D}|$ has a right Kan extension along the projection $\pi : |\mathbf{D}|^{n+1} \to |\mathbf{D}|^n$ of the first n coordinates. The Kan extension is given by

$$\begin{aligned}(\mathrm{RK}_\pi(f^p, f^r))^p(\vec{R}) = \{(x, y) \mid{}& \forall S : |\mathbf{CCP}|.\, f^p(\vec{R}, S)(x, y) \wedge \\ & \forall S, S' : |\mathbf{CCP}|.\, \forall A : \mathrm{AdmRel}_{\mathbf{CCP}}(S, S').\, f^r(\vec{eq}_{\vec{R}}, A)([x], [y])\} \\ (\mathrm{RK}_\pi(f^p, f^r))^r(\vec{A}) = \{([x], [y]) \mid{}& \forall S, S' : |\mathbf{CCP}|.\, \forall A' : \mathrm{AdmRel}_{\mathbf{CCP}}(S, S'). \\ & f^r(\vec{A}, A')([x], [y])\}\end{aligned}$$

and showing that this works is quite standard (see, for example, Jacobs (1999, Chapter 8) or Birkedal and Møgelberg (2005)). Since Definition 4.7 only calls for the existence of Kan extensions of functors obtained by composing with lifting, this construction gives us all the Kan extensions we need. For the Beck–Chevalley condition, suppose $(g^p, g^r) : |\mathbf{D}|^m \to |\mathbf{D}|^n$. Then, clearly, $\mathrm{RK}_\pi((f^p, f^r) \circ ((g^p, g^r) \times id_{|\mathbf{D}|})) = \mathrm{RK}_\pi((f^p, f^r)) \circ (g^p, g^r)$ and the map of the Beck–Chevalley condition, which is required to be an isomorphism, is in fact an identity.

To show that the per-model models full PolyFPC, we need to show that the collection of solvable recursive domain equations is closed under Kleisli polymorphism. Birkedal *et al.* (2007) showed that the class of solvable recursive domain equations in the PILL_Y model $\mathbf{C}$ is exactly the collection of internal functors in $\mathrm{RT}(D)^G$. This means that the collection of solvable recursive domain equations in the FPC model $(\mathbf{C}, L)$ is the collection of functors $(\mathbf{C}_L{}^{\mathrm{op}} \times \mathbf{C}_L)^{n+1} \to \mathbf{C}_L$ induced by internal functors $(f^p, f^r) : (\mathbf{C}^{\mathrm{op}} \times \mathbf{C})^{n+1} \to \mathbf{C}$ and $(g^p, g^r) : (\mathbf{D}^{\mathrm{op}} \times \mathbf{D})^{n+1} \to \mathbf{D}$ such that (g^p, g^r) lifts (f^p, f^r). The latter means that the diagrams

$$\begin{tikzcd}
(\mathbf{CCP}^{\mathrm{op}} \times \mathbf{CCP})^{n+1} \arrow[r, "f^p"] \arrow[d, "(L^{\mathrm{op}} \times L)^{n+1}"'] & \mathbf{CCP} \arrow[d, "L"] \\
(\mathbf{AP}_\perp{}^{\mathrm{op}} \times \mathbf{AP}_\perp)^{n+1} \arrow[r, "g^p"] & \mathbf{AP}_\perp
\end{tikzcd}$$

$$\begin{tikzcd}
(\mathbf{AdmRel}_{\mathbf{CCP}}{}^{\mathrm{op}} \times \mathbf{AdmRel}_{\mathbf{CCP}})^{n+1} \arrow[r, "f^r"] \arrow[d, "(L^{\mathrm{op}} \times L)^{n+1}"'] & \mathbf{AdmRel}_{\mathbf{CCP}} \arrow[d, "L"] \\
(\mathbf{AdmRelAP}_\perp{}^{\mathrm{op}} \times \mathbf{AdmRelAP}_\perp)^{n+1} \arrow[r, "g^r"] & \mathbf{AdmRelAP}_\perp
\end{tikzcd}$$

commute (using the fact that the composition of the equivalence $\mathbf{CCP} \to \mathbf{AP}_\perp^L$ and the forgetful functor $\mathbf{AP}_\perp^L : \mathbf{AP}_\perp$ is lifting, as noted earlier). Consider $\tilde{\prod} f = ((\tilde{\prod} f)^p, (\tilde{\prod} f)^r)$ and $\tilde{\prod} g = ((\tilde{\prod} g)^p, (\tilde{\prod} g)^r)$ defined as follows:

$$\begin{aligned}
(\tilde{\prod} f)^p(\vec{R}) = L\{(x, y) \mid \forall S : |\mathbf{CCP}|.\, f^p(\vec{R}, S, S)(x, y) \wedge \\
\forall S, S' : |\mathbf{CCP}|.\, \forall A : \mathrm{AdmRel}_{\mathbf{CCP}}(S, S').\, f^r(\vec{eq}_{\vec{R}}, A, A)([x], [y])\} \\
\tilde{\prod} f^r(\vec{A}) = L\{([x], [y]) \mid \forall S, S' : |\mathbf{CCP}|.\, \forall A' : \mathrm{AdmRel}_{\mathbf{CCP}}(S, S'). \\
g^r(\vec{A}, A', A')([x], [y])\} \\
\tilde{\prod} g^p(\vec{R}) = L\{(x, y) \mid \forall S : |\mathbf{CCP}|.\, g^p(\vec{R}, LS, LS)(x, y) \wedge \\
\forall S, S' : |\mathbf{CCP}|.\, \forall A : \mathrm{AdmRel}_{\mathbf{CCP}}(S, S').\, g^r(\vec{eq}_{\vec{R}}, LA, LA)([x], [y])\} \\
\tilde{\prod} g^r(\vec{A}) = L\{([x], [y]) \mid \forall S, S' : |\mathbf{CCP}|.\, \forall A' : \mathrm{AdmRel}_{\mathbf{CCP}}(S, S'). \\
g^r(\vec{A}, LA', LA')([x], [y])\}.
\end{aligned}$$

One can verify that $\tilde{\prod} g$ lifts $\tilde{\prod} f$, and that the functor $(\tilde{\prod} f, \tilde{\prod} g) : (\mathbf{C}_L{}^{\mathrm{op}} \times \mathbf{C}_L)^n \to \mathbf{C}_L$ induced by the two is the one constructed from the Kan extension using Lemma 4.12. This shows that the class of solvable recursive domain equations is closed under Kleisli polymorphism, as required. □

7.1. *The per-model of PolyFPC*

We now describe the resulting interpretation of PolyFPC in the per-model. Types of PolyFPC are modelled as pairs $([\![\vec{\alpha} \vdash \sigma]\!]^p, [\![\vec{\alpha} \vdash \sigma]\!]^r)$, where $[\![\vec{\alpha} \vdash \sigma]\!]^p$ is a map $|\mathbf{CCP}|^n \to |\mathbf{CCP}|$ and $[\![\vec{\alpha} \vdash \sigma]\!]^r$ is a map taking an n-vector $(A_i : \mathbf{AdmRel}(R_i, S_i))$ of admissible relations (admissible in the sense of objects of $\mathbf{AdmRel}_{\mathbf{CCP}}$) on objects of $\mathbf{CCP}$ and produces an admissible relation

$$[\![\vec{\alpha} \vdash \sigma]\!]^r(\vec{A}) : \mathbf{AdmRel}([\![\vec{\alpha} \vdash \sigma]\!]^p(\vec{R}), [\![\vec{\alpha} \vdash \sigma]\!]^p(\vec{S}))$$

satisfying $[\![\vec{\alpha} \vdash \sigma]\!]^r(eq_{\vec{R}}) = eq_{[\![\vec{\alpha} \vdash \sigma]\!]^p(\vec{R})}$.

Figure 6 shows the interpretation of all the PolyFPC types in the per-model except the recursive types. These recursive types are determined uniquely up to isomorphism by their

$$\begin{aligned}
[\![\vec{\alpha} \vdash \alpha_i]\!]^p(\vec{R}) &= R_i \\
[\![\vec{\alpha} \vdash \sigma \times \tau]\!]^p(\vec{R}) &= \{(\langle x, y\rangle, \langle x', y'\rangle \mid [\![\vec{\alpha} \vdash \sigma]\!]^p(\vec{R})(x, x') \wedge [\![\vec{\alpha} \vdash \tau]\!]^p(\vec{R})(y, y')\} \\
[\![\vec{\alpha} \vdash \sigma + \tau]\!]^p(\vec{R}) &= \{(\langle 1, x\rangle, \langle 1, x'\rangle) \mid [\![\vec{\alpha} \vdash \sigma]\!]^p(\vec{R})(x, x')\} \cup \\
&\quad\ \{(\langle 2, y\rangle, \langle 2, y'\rangle) \mid [\![\vec{\alpha} \vdash \tau]\!]^p(\vec{R})(y, y')\} \\
[\![\vec{\alpha} \vdash \sigma \to \tau]\!]^p(\vec{R}) &= \{(e, f) \mid \forall x, y \in D.\, [\![\vec{\alpha} \vdash \sigma]\!]^p(\vec{R})(x, y) \supset L[\![\vec{\alpha} \vdash \tau]\!]^p(\vec{R})(e \cdot x, f \cdot y)\} \\
[\![\vec{\alpha} \vdash 1]\!]^p(\vec{R}) &= \{(\bot, \bot)\} \\
[\![\vec{\alpha} \vdash \textstyle\prod \alpha.\, \sigma]\!]^p(\vec{R}) &= \{(x, y) \mid \forall S \colon |\mathbf{CCP}|.\, L[\![\vec{\alpha}, \alpha \vdash \sigma]\!]^p(\vec{R}, S)(x, y) \wedge \\
&\quad\ \forall S, S' \colon |\mathbf{CCP}|.\, \forall A \colon \mathrm{AdmRel}_{\mathbf{CCP}}(S, S').\, L[\![\vec{\alpha}, \alpha \vdash \sigma]\!]^r(\vec{eq}_{\vec{R}}, A)([x], [y])\}
\end{aligned}$$

$$\begin{aligned}
[\![\vec{\alpha} \vdash \alpha_i]\!]^r(\vec{A}) &= A_i \\
[\![\vec{\alpha} \vdash \sigma \times \tau]\!]^r(\vec{A}) &= \{([\langle x, y\rangle], [\langle x', y'\rangle]) \mid [\![\vec{\alpha} \vdash \sigma]\!]^r(\vec{A})([x], [x']) \wedge [\![\vec{\alpha} \vdash \tau]\!]^r(\vec{A})([y], [y'])\} \\
[\![\vec{\alpha} \vdash \sigma + \tau]\!]^r(\vec{A}) &= \{([\langle 1, x\rangle], [\langle 1, x'\rangle]) \mid [\![\vec{\alpha} \vdash \sigma]\!]^r(\vec{A})([x], [x'])\} \cup \\
&\quad\ \{([\langle 2, y\rangle], [\langle 2, y'\rangle]) \mid [\![\vec{\alpha} \vdash \tau]\!]^r(\vec{A})([y], [y'])\} \\
[\![\vec{\alpha} \vdash \sigma \to \tau]\!]^r(\vec{A}) &= \{([e], [f]) \mid \forall ([x], [y]) \in [\![\vec{\alpha} \vdash \sigma]\!]^r(\vec{A}).\, ([e \cdot x], [f \cdot y]) \in L[\![\vec{\alpha} \vdash \tau]\!]^r(\vec{A})\} \\
[\![\vec{\alpha} \vdash 1]\!]^r(\vec{A}) &= \{([\bot], [\bot])\} \\
[\![\vec{\alpha} \vdash \textstyle\prod \alpha.\, \sigma]\!]^r(\vec{A}) &= \{([x], [y]) \mid \forall S, S' \colon |\mathbf{CCP}|.\, \forall A \colon \mathrm{AdmRel}_{\mathbf{CCP}}(S, S'). \\
&\quad\ L[\![\vec{\alpha}, \alpha \vdash \sigma]\!]^r(\vec{A}, A)([x], [y])\}
\end{aligned}$$

Fig. 6. Interpretation of PolyFPC in per-model

universal properties (as initial dialgebras), a property shown to be useful in Pitts (1996), and we shall not present a more concrete description of them in this model. In the figure, $\langle\cdot,\cdot\rangle$ denotes the pairing function $D \times D \to D$ definable using the combinatory algebra structure on D, and the symbols 1 and 2 denote incomparable elements of D (explicitly, these could be $\langle \iota, \bot\rangle$ and $\langle \bot, \iota\rangle$, respectively, where ι denotes a code for the identity function on D).

The monad induced by the comonad on $\mathbf{AP}_\bot$ and $\mathbf{AdmRel}_{\mathbf{AP}_\bot}$ is denoted L. Explicitly, this monad maps a chain complete per R to $\{(\bot, \bot)\} \cup \{(\langle \iota, x\rangle, \langle \iota, y\rangle) \mid R(x, y)\}$, and an admissible relation A on chain complete pers R, S is mapped to the relation on LR, LS that relates $[\bot]$ to $[\bot]$ and $[\langle \iota, x\rangle]$ to $[\langle \iota, y\rangle]$ if $A([x], [y])$.

Terms of PolyFPC are modelled in the Kleisli category for L. To be more precise, a term $\vec{\alpha} \mid \vec{x} : \vec{\sigma} \vdash t : \tau$ is modelled as an indexed family of maps

$$([\![t]\!]_{\vec{R}} : \textstyle\prod_i [\![\vec{\alpha} \vdash \sigma_i]\!]^p(\vec{R}) \to L[\![\vec{\alpha} \vdash \tau]\!]^p(\vec{R}))_{\vec{R}}$$

where the product refers to the product in **CCP**. Such a family must have a common tracker, and must preserve relations, which means that if $\vec{A} : \mathbf{AdmRel}(\vec{R}, \vec{S})$, and for each i, $([x_i], [y_i]) \in [\![\vec{\alpha} \vdash \sigma_i]\!]^r(\vec{A})$, then

$$([\![t]\!]_{\vec{R}}([x_1], \ldots, [x_m]), [\![t]\!]_{\vec{S}}([y_1], \ldots, [y_m])) \in L[\![\vec{\alpha} \vdash \tau]\!]^r(\vec{A}).$$

Theorem 7.6. The interpretation of PolyFPC types defined in Figure 6 extends to a sound interpretation of PolyFPC.

7.2. *Computational adequacy*

A τ-σ context of PolyFPC for types σ, τ, where σ is closed, is an expression $C[-]$ containing a place holder $-$ such that whenever an expression t of type τ is substituted for the place holder such that the result $C[t]$ is a closed term, it has type σ. Two terms $t, t' : \tau$ of PolyFPC of the same type are called contextually equivalent (written $t \equiv t'$), if for any type σ and any τ-σ context $C[-]$,

$$C[t] \Downarrow \text{ iff } C[t'] \Downarrow$$

where $t \Downarrow$ means that there exists a v such that $t \Downarrow v$.

Theorem 7.7 (Adequacy). For any program t of PolyFPC, $[\![t]\!] \neq [\bot]$ in the per-model if and only if $t \Downarrow$.

The following corollary giving a tight connection between the operational and denotational semantics is easily proved from Theorem 7.7.

Corollary 7.8. Suppose t, t' are two PolyFPC terms of the same type. If $[\![t]\!] = [\![t']\!]$, then $t \equiv t'$.

Proof. Since the interpretation of terms is defined by structural induction, one may easily prove that for any τ-σ context $C[-]$, and terms t, t' of type τ, if $[\![t]\!] = [\![t']\!]$, then $[\![C[t]]\!] = [\![C[t']]\!]$. So, if $[\![t]\!] = [\![t']\!]$, then, for any context $C[-]$,

$$C[t] \Downarrow \iff [\![C[t]]\!] \neq [\bot] \iff [\![C[t']]\!] \neq [\bot] \iff C[t'] \Downarrow,$$

which proves the corollary. □

One can define a contextual equivalence relation on FPC as the contextual equivalence for PolyFPC by restricting to the contexts of FPC. Since this equivalence relation is obtained by quantifying over fewer contexts, it is weaker than the contextual equivalence relation inherited from PolyFPC by restriction. Corollary 7.8 now gives a computational adequacy result of the interpretation of FPC into PILL$_Y$ with respect to this contextual equivalence relation.

Corollary 7.9. The interpretation $(-)^*$ of FPC into PILL$_Y$ of Section 6 is computationally adequate in the sense that for two FPC programs t, t' of the same type, if t^* is provably equal to t'^* in LAPL using parametricity, then t and t' are contextually equivalent.

Proof. The restriction to FPC of the interpretation of PolyFPC into the per model can be factorised as $(-)^*$ followed by the interpretation of PILL$_Y$ into the per-model. This means that if t^* is provably equal to t'^*, then, by soundness of the per-model interpretation of Linear Abadi & Plotkin Logic, $[\![t]\!] = [\![t']\!]$, so the corollary then follows from Corollary 7.8. □

We now proceed to prove Theorem 7.7.

In the following, we shall refer to admissible relations between chain complete pers R and sets Y, writing $\mathrm{AdmRel}_{\mathbf{CCP}}(R, Y)$ for the collection of all such. By this, we mean subsets A of $(D/R) \times Y$ such that for each $y \in Y$, the collection $\{[x]_R \mid ([x]_R, y) \in A\}$ defines a chain complete per.

For the proof of Theorem 7.7, we construct for each *open* type of PolyFPC an approximation relation $\preccurlyeq_\sigma$, which for each vector of closed PolyFPC-types $\vec{\tau}$ and vector of admissible relations $(\rho_i : \mathrm{AdmRel}_{\mathbf{CCP}}([\![\tau_i]\!]^p, \mathrm{Val}_{\tau_i}))_i$ gives an admissible relation

$$\preccurlyeq_\sigma^{(\vec{\tau},\vec{\rho})} : \mathrm{AdmRel}_{\mathbf{CCP}}([\![\sigma]\!]^p([\![\vec{\tau}]\!]^p), \mathrm{Val}_{\sigma[\vec{\tau}/\vec{\alpha}]}),$$

where Val_τ denotes the set of values of PolyFPC of type τ.

In the following, when we use notation such as $x : [\![\sigma]\!]^p$ or $x : R$ for pers R or $x \preccurlyeq_\sigma^{(\vec{\tau},\vec{\rho})} v$, the variable x ranges over the collection of equivalence classes for the per in question. This follows the intuition mentioned above in which equivalence classes are the elements of a per. If $x : [\![\sigma \to \tau]\!]^p$ and $d : [\![\sigma]\!]^p$, we shall use the notation $x(d) : L[\![\tau]\!]^p$ for the equivalence class obtained by applying any representative of x to any representative of d. This is well defined by the definition of $[\![\sigma \to \tau]\!]^p$. Similarly, if $x : [\![\prod \alpha.\, \sigma]\!]^p$ and R is any chain complete per, we shall write $x(R) : L([\![\alpha \vdash \sigma]\!]^p(R))$ for the equivalence class represented by any representative of x. If $x : LR$, we shall write $x\downarrow$ for $x \neq [\bot]_{LR}$. Finally, we shall write Type for the set of closed PolyFPC types.

Lemma 7.10. The approximation relations indexed over PolyFPC types can be defined such that:

— $x \preccurlyeq_{\alpha_i}^{(\vec{\tau},\vec{\rho})} v \iff x \rho_i v$

— $x \preccurlyeq_{1}^{(\vec{\tau},\vec{\rho})} * \iff \top$

— $x \preccurlyeq_{\sigma\times\tau}^{(\vec{\tau},\vec{\rho})} \langle v, v' \rangle \iff \pi_1 x \preccurlyeq_{\sigma}^{(\vec{\tau},\vec{\rho})} v \wedge \pi_2 x \preccurlyeq_{\tau}^{(\vec{\tau},\vec{\rho})} v'$

— $x \preccurlyeq_{\sigma+\tau}^{(\vec{\tau},\vec{\rho})} \mathtt{inl}\, v \iff \exists x'.\, x = \mathit{inl}\,(x') \wedge x' \preccurlyeq_{\sigma}^{(\vec{\tau},\vec{\rho})} v$

— $x \preccurlyeq_{\sigma+\tau}^{(\vec{\tau},\vec{\rho})} \mathtt{inr}\, v \iff \exists x'.\, x = \mathit{inr}\,(x') \wedge x' \preccurlyeq_{\tau}^{(\vec{\tau},\vec{\rho})} v$

— $x \preccurlyeq_{\sigma\to\tau}^{(\vec{\tau},\vec{\rho})} \lambda y : \sigma.\, t \iff \forall d, v'.\, (d \preccurlyeq_{\sigma}^{(\vec{\tau},\vec{\rho})} v' \wedge x(d)\downarrow) \supset \exists v''.\, t[v'/y] \Downarrow v'' \wedge x(d) \preccurlyeq_{\tau}^{(\vec{\tau},\vec{\rho})} v''$

— $x \preccurlyeq_{\prod \alpha.\sigma}^{(\vec{\tau},\vec{\rho})} \Lambda\alpha.\, t \iff \forall \tau : \mathsf{Type}.\, \forall \rho : \mathrm{AdmRel}_{\mathbf{CCP}}([\![\tau]\!]^p, \mathrm{Val}_\tau).\, x([\![\tau]\!]^p) \downarrow\supset \exists v'.\, t[\tau/\alpha] \Downarrow v' \wedge x([\![\tau]\!]^p) \preccurlyeq_{\sigma}^{(\vec{\tau},\tau,\vec{\rho},\rho)} v'$.

— $x \preccurlyeq_{\mathrm{rec}\ \alpha.\sigma}^{(\vec{\tau},\vec{\rho})} \mathtt{intro}\, v \iff \mathit{unfold}(x) \preccurlyeq_{\sigma}^{(\vec{\tau},\mathrm{rec}\ \alpha.\sigma[\vec{\tau}/\vec{\alpha}],\vec{\rho},\preccurlyeq_{\mathrm{rec}\ \alpha.\sigma}^{(\vec{\tau},\vec{\rho})})} v$

where *inl*, *inr* denote, respectively, the left and right inclusions into the coproduct, and in the last condition, *unfold* is the isomorphism $[\![\mathrm{rec}\ \alpha.\, \sigma]\!]^p \to [\![\sigma[\mathrm{rec}\ \alpha.\, \sigma/\alpha]]\!]^p$ (that is, the interpretation of `elim`).

Lemma 7.10 may at first appear to be a definition of $\preccurlyeq_\sigma$, but since the case of recursive types involves the relation being defined on the right-hand side of the equation, the proof of the lemma below involves finding a fixed point for a certain map of predicates, as in Pitts (1996).

Lemma 7.11. For any term

$$\vec{\alpha} \mid \vec{x} : \vec{\sigma} \vdash t : \tau$$

of PolyFPC, for all vectors of closed types $\vec{\tau}$ and all vectors of relations

$$(\rho_i : \mathrm{AdmRel}_{\mathbf{CCP}}(\llbracket \tau_i \rrbracket^p, \mathrm{Val}_{\tau_i}))_i$$

and all $(d_i : \llbracket \sigma_i[\vec{\tau}/\vec{\alpha}] \rrbracket^p))_i, (v_i : \mathrm{Val}_{\sigma_i[\vec{\tau}/\vec{\alpha}]})_i$, if for all i

$$d_i \preccurlyeq_{\sigma_i}^{(\vec{\tau},\vec{\rho})} v_i,$$

then $\llbracket t \rrbracket_{\llbracket \vec{\tau} \rrbracket^p}(\vec{d}) \downarrow$ implies

$$\exists v : \mathrm{Val}.\, t[\vec{\tau}/\vec{\alpha}][\vec{v}/\vec{x}] \Downarrow v \wedge \llbracket t \rrbracket_{\llbracket \vec{\tau} \rrbracket^p}(\vec{d}) \preccurlyeq_{\tau}^{(\vec{\tau},\vec{\rho})} v.$$

Assuming Lemma 7.10 and Lemma 7.11 are correct (proofs follow below), Theorem 7.7 is now easy.

Proof of Theorem 7.7. The 'only if' direction is the special case of Lemma 7.11 for closed terms.

For the 'if' direction, suppose $t : \tau$ and $t \Downarrow v$. By soundness, $\llbracket t \rrbracket = \llbracket v \rrbracket$, and by Lemma 4.15, $\llbracket v \rrbracket$ considered as a map from 1 to $\llbracket \tau \rrbracket$ is a map of coalgebras, meaning in this case that it is total, that is, $\llbracket v \rrbracket \neq \perp$. □

We now proceed to prove first Lemma 7.10 and then Lemma 7.11. Recall that each PolyFPC type $\sigma(\alpha, \beta)$ in which α occurs only negatively and β only positively induces a functor $\sigma : \mathbf{CCP}_L{}^{\mathrm{op}} \times \mathbf{CCP}_L \to \mathbf{CCP}_L$. In the following, for R, S chain complete pers, A, B admissible predicates on R, S, respectively, in the usual sense, and $f : R \rightharpoonup S$ in $\mathbf{CCP}_L$, the notation $f : A \rightharpoonup B$ means for all $x \in A$, $f(x) \downarrow$ implies $f(x) \in B$. In fact, $A \rightharpoonup B$ defines an admissible predicate on the chain complete per $R \rightharpoonup S$.

The next lemma is a consequence of a more general induction/coinduction principle for recursive types, as in Pitts (1996).

Lemma 7.12. Suppose $\sigma(\alpha, \beta)$ is a PolyFPC type in which α occurs only negatively and β only positively, and A, B are admissible predicates on $\llbracket \mathrm{rec}\ \alpha.\, \sigma(\alpha, \alpha) \rrbracket^p$ in the usual sense. Suppose also that for all maps $e : \llbracket \mathrm{rec}\ \alpha.\, \sigma(\alpha, \alpha) \rrbracket^p \rightharpoonup \llbracket \mathrm{rec}\ \alpha.\, \sigma(\alpha, \alpha) \rrbracket^p$ in $\mathbf{CCP}_L$, if $e : A \rightharpoonup B$, then $\mathit{fold} \circ \sigma(e, e) \circ \mathit{unfold} : A \rightharpoonup B$, where

$$\mathit{unfold} : \llbracket \mathrm{rec}\ \alpha.\, \sigma(\alpha, \alpha) \rrbracket^p \to \llbracket \sigma(\mathrm{rec}\ \alpha.\, \sigma(\alpha, \alpha), \mathrm{rec}\ \alpha.\, \sigma(\alpha, \alpha)) \rrbracket^p$$

is the isomorphism and $\mathit{fold} = \mathit{unfold}^{-1}$. Then $A \subseteq B$.

Proof. Consider the map

$$\begin{aligned} e \mapsto \mathit{fold} \circ \sigma(e, e) \circ \mathit{unfold} :& \\ (\llbracket \mathrm{rec}\ \alpha.\, \sigma(\alpha, \alpha) \rrbracket^p \rightharpoonup \llbracket \mathrm{rec}\ \alpha.\, \sigma(\alpha, \alpha) \rrbracket^p) &\to \\ (\llbracket \mathrm{rec}\ \alpha.\, \sigma(\alpha, \alpha) \rrbracket^p \rightharpoonup \llbracket \mathrm{rec}\ \alpha.\, \sigma(\alpha, \alpha) \rrbracket^p)&. \end{aligned} \tag{10}$$

Since $\mathbf{CCP}_L$ is equivalent to a subcategory of $\mathbf{AP}_\perp$ (namely, the full subcategory of objects of the form $L(R)$ for R a chain complete per), this map must have a fixed point, since all endomaps in $\mathbf{AP}_\perp$ have fixed points. Moreover, since $A \rightharpoonup B$ is an admissible predicate, it corresponds to an admissible predicate on the corresponding object in $\mathbf{AP}_\perp$, and thus, by fixed-point induction in $\mathbf{AP}_\perp$, the fixed point of (10) satisfies $A \rightharpoonup B$.

Now, dinaturality of the solutions to recursive domain equations implies that the identity is the only fixed point of (10), and thus $A \subseteq B$, as required. □

Lemma 7.10 will be proved by synchronous induction on the structure of σ with the following Lemma.

Lemma 7.13. Suppose $\sigma(\vec{\alpha}, \alpha, \beta)$ is a type of PolyFPC in which α occurs only negatively and β only positively. Suppose also that we are given closed types $\vec{\tau}, \tau_-, \tau_+$ and relations

$$\vec{\rho} : \mathrm{AdmRel}_{\mathbf{CCP}}([\![\vec{\tau}]\!]^p, \mathrm{Val}_{\vec{\tau}})$$

$$\rho_-, \rho'_- : \mathrm{AdmRel}_{\mathbf{CCP}}([\![\tau_-]\!]^p, \mathrm{Val}_{\tau_-})$$

$$\rho_+, \rho'_+ : \mathrm{AdmRel}_{\mathbf{CCP}}([\![\tau_+]\!]^p, \mathrm{Val}_{\tau_+})$$

and maps $e_- : [\![\tau_-]\!]^p \rightharpoonup [\![\tau_-]\!]^p$, $e_+ : [\![\tau_+]\!]^p \rightharpoonup [\![\tau_+]\!]^p$ in the model . If $e_- : \rho'_- \rightharpoonup \rho_-$ and $e_+ : \rho_+ \rightharpoonup \rho'_+$ in the sense that

$$\forall x, v. (x, v) \in \rho'_- \wedge e_-(x) \downarrow \supset (e_-(x), v) \in \rho_-$$
$$\forall x, v. (x, v) \in \rho_+ \wedge e_+(x) \downarrow \supset (e_+(x), v) \in \rho'_+,$$

then

$$\sigma(\vec{\tau}, e_-, e_+) : \ \preccurlyeq_\sigma^{\vec{\tau},\tau_-,\tau_+,\vec{\rho},\rho_-,\rho_+} \rightharpoonup \preccurlyeq_\sigma^{\vec{\tau},\tau_-,\tau_+,\vec{\rho},\rho'_-,\rho'_+}$$

in the same sense

Proof of Lemma 7.10 and 7.13. As we have already said, these are proved by synchronous structural induction over σ.

We start with Lemma 7.10. The conditions listed in the lemma basically give an inductive definition, except for the case of recursive types. For this case we adapt the proof from Pitts (1996).

We will assume that we have split the occurrences of α in σ into positive and negative, that is, that we are working with a type $\sigma(\vec{\alpha}, \alpha, \beta)$ where α occurs only negatively and β only positively. We will write rec $\alpha. \sigma$ to mean $\vec{\alpha} \vdash \mathrm{rec}\ \alpha. \sigma(\vec{\alpha}, \alpha, \alpha)$. To aid readability of the following lines, we will write A for

$$\mathrm{AdmRel}_{\mathbf{CCP}}([\![\mathrm{rec}\ \alpha. \sigma]\!]^p([\![\vec{\tau}]\!]^p), \mathrm{Val}_{\mathrm{rec}\ \alpha.\sigma[\vec{\tau}/\vec{\alpha}]}).$$

Consider the map $\Psi : A \times A \to A$ defined to map (ρ_-, ρ_+) to

$$(x, \mathtt{intro}\ v). \mathit{unfold}(x) \preccurlyeq_\sigma^{(\vec{\tau}, \mathrm{rec}\ \alpha.\sigma[\vec{\tau}/\vec{\alpha}], \mathrm{rec}\ \alpha.\sigma[\vec{\tau}/\vec{\alpha}], \vec{\rho}, \rho_-, \rho_+)} v.$$

By Lemma 7.13 (or one could say, the Lemma 7.13 part of the induction hypothesis), this is actually a functor

$$\Psi : A^{\mathrm{op}} \times A \to A$$

with respect to the ordering on relations.

We need to construct a Δ such that $\Psi(\Delta, \Delta) = \Delta$. Define $\Phi : A^{\mathrm{op}} \times A \to A^{\mathrm{op}} \times A$ by $\Phi(\rho_-, \rho_+) = (\Psi(\rho_+, \rho_-), \Psi(\rho_-, \rho_+))$. Since A is closed under intersections, it is a complete lattice, and hence $A^{\mathrm{op}} \times A$ is a complete lattice also. This means that since Φ preserves the ordering, the Tarski–Knaster theorem states that there exists a least fixed point (Δ_-, Δ_+) for Φ. We aim to show that $\Delta_- = \Delta_+$.

By the symmetry of the definition of Φ, we have $\Phi(\Delta_+, \Delta_-) = (\Delta_+, \Delta_-)$, which implies $(\Delta_-, \Delta_+) \leqslant (\Delta_+, \Delta_-)$, meaning $\Delta_+ \leqslant \Delta_-$ since (Δ_-, Δ_+) is the least fixed point.

Finally, we need to show that $\Delta_- \leqslant \Delta_+$. By Lemma 7.12, it suffices to show that if $e \colon \Delta_- \rightharpoonup \Delta_+$, then $\mathit{fold} \circ \sigma(e,e) \circ \mathit{unfold} \colon \Delta_- \rightharpoonup \Delta_+$ also. So suppose $e \colon \Delta_- \rightharpoonup \Delta_+$ and $(x, \mathtt{intro}\, v) \in \Delta_-$. Since $\Delta_- = \Psi(\Delta_+, \Delta_-)$, this means that

$$\mathit{unfold}(x) \preccurlyeq_{\sigma}^{(\vec{\tau}, \text{rec } \alpha.\sigma[\vec{\tau}/\vec{\alpha}], \text{rec } \alpha.\sigma[\vec{\tau}/\vec{\alpha}], \vec{\rho}, \Delta_+, \Delta_-)} v.$$

By Lemma 7.13, if $\sigma(e,e) \circ \mathit{unfold}(x) \downarrow$, then

$$\sigma(e,e) \circ \mathit{unfold}(x) \preccurlyeq_{\sigma}^{(\vec{\tau}, \text{rec } \alpha.\sigma[\vec{\tau}/\vec{\alpha}], \text{rec } \alpha.\sigma[\vec{\tau}/\vec{\alpha}], \vec{\rho}, \Delta_-, \Delta_+)} v,$$

which, since $\Delta_+ = \Psi(\Delta_-, \Delta_+)$, implies $\Delta_+(\mathit{fold} \circ \sigma(e,e) \circ \mathit{unfold}(x), \mathtt{intro}\, v)$.

Now we will prove Lemma 7.13. As we have already said, the proof is by induction on σ, but we have just proved the cases of the type constructors $\to$ and recursive types. For the case of $\sigma \to \sigma'$, suppose $x \preccurlyeq_{\sigma \to \sigma'}^{(\vec{\tau}, \tau_-, \tau_+, \rho_-, \rho_+)} \lambda y.\, t$. If $(\sigma \to \sigma')(id_{\vec{\tau}}, e_-, e_+)(x) \downarrow$, we must show that

$$(\sigma \to \sigma')(id_{\vec{\tau}}, e_-, e_+)(x) \preccurlyeq_{\sigma \to \sigma'}^{(\vec{\tau}, \tau_-, \tau_+, \rho'_-, \rho'_+)} \lambda y.\, t.$$

By definition,

$$(\sigma \to \sigma')(id_{\vec{\tau}}, e_-, e_+)(x) = \sigma(id_{\vec{\tau}}, e_-, e_+) \circ x \circ \sigma'(id_{\vec{\tau}}, e_+, e_-).$$

Assume also that we are given d, v such that $d \preccurlyeq_{\sigma}^{(\vec{\tau}, \tau_+, \tau_-, \rho'_+, \rho'_-)} v$ (where we have swapped appearances of $+$ and $-$ since $(\sigma \to \sigma')(\vec{\tau}, \tau_-, \tau_+) = \sigma(\vec{\tau}, \tau_+, \tau_-) \to \sigma'(\vec{\tau}, \tau_-, \tau_+)$), and $(\sigma \to \sigma')(id_{\vec{\tau}}, e_-, e_+)(x)(d) \downarrow$. By the induction hypothesis on σ', we have

$$\sigma'(id_{\vec{\tau}}, e_+, e_-)(d) \preccurlyeq_{\sigma'}^{(\vec{\tau}, \tau_+, \tau_-, \rho_+, \rho_-)} v.$$

So, since $x(\sigma'(id_{\vec{\tau}}, e_+, e_-)(d)) \downarrow$, there exists a v' such that $t[v/y] \Downarrow v'$ and

$$x(\sigma'(id_{\vec{\tau}}, e_+, e_-)(d)) \preccurlyeq_{\sigma}^{(\vec{\tau}, \tau_-, \tau_+, \rho_-, \rho_+)} v'.$$

The rest of the proof now follows from the induction hypothesis for σ.

For the recursive types, the pairs (ρ_-, ρ_+) and (ρ_-, ρ_+) define functors Φ and Φ', respectively, for which $\preccurlyeq_{\text{rec } \alpha.\sigma}^{(\vec{\tau}, \tau_-, \tau_+, \rho, \rho_-, \rho_+)}$ and $\preccurlyeq_{\text{rec } \alpha.\sigma}^{(\vec{\tau}, \tau_-, \tau_+, \rho, \rho_-, \rho_+)}$, respectively, are defined to be least fixed points. By the induction hypothesis, rec $\alpha.\, \sigma(e_-, e_+)$ defines a natural transformation between the two, and hence also a map between the fixed points. □

Lemma 7.14. For any pair of PolyFPC types $\vec{\alpha}, \alpha \vdash \sigma$ and $\vec{\alpha} \vdash \omega$, any $\vec{\tau}, \vec{\rho}$,

$$\preccurlyeq_{\sigma}^{(\vec{\tau}, \omega[\vec{\tau}/\vec{\alpha}], \vec{\rho}, \preccurlyeq_{\omega}^{\vec{\tau}, \vec{\rho}})} \iff \preccurlyeq_{\sigma[\omega/\alpha]}^{\vec{\tau}, \vec{\rho}}.$$

As a consequence,

$$x \preccurlyeq_{\text{rec } \alpha.\sigma}^{(\vec{\tau}, \vec{\rho})} \mathtt{intro}\,(v) \iff \mathit{unfold}(x) \preccurlyeq_{\sigma[\text{rec } \alpha.\sigma/\alpha]}^{(\vec{\tau}, \vec{\rho})} v.$$

Proof. The proof is an easy induction on σ. □

Proof of Lemma 7.11. The proof is by induction on the structure of the term t:

$t = x_i$.

This case is trivial.

$t = \mathtt{inl}\,(t')$.

Here $\tau = \tau' + \tau''$. Suppose $[\![t]\!]_{[\![\vec{\tau}]\!]^p}(\vec{d}) \downarrow$. Then $[\![t']\!]_{[\![\vec{\tau}]\!]^p}(\vec{d}) \downarrow$ also, so, by induction, there exists v such that $t'[\vec{\tau}/\vec{\alpha}] \Downarrow v$ and $[\![t']\!]_{[\![\vec{\tau}]\!]^p}(\vec{d}) \preccurlyeq_{\tau'}^{(\vec{\tau},\vec{\rho})} v$. This implies that $t[\vec{\tau}/\vec{\alpha}] \Downarrow \mathtt{inl}\, v$ and $[\![t]\!]_{[\![\vec{\tau}]\!]^p}(\vec{d}) \preccurlyeq_{\tau'+\tau''}^{(\vec{\tau},\vec{\rho})} \mathtt{inl}\,(v)$.

$t = \mathtt{case}\; t' \;\mathtt{of}\; \mathtt{inl}\; x.\, t'' \;\mathtt{of}\; \mathtt{inr}\; x.\, t'''$.

Suppose the whole expression has type τ, and t' has type $\tau' + \tau''$. Suppose $[\![t]\!]_{[\![\vec{\tau}]\!]^p}(\vec{d}) \downarrow$. Then $[\![t']\!]_{[\![\vec{\tau}]\!]^p}(\vec{d}) \downarrow$ also, so, by the induction hypothesis, there exists a v' such that $t'[\vec{\tau}/\vec{\alpha}][\vec{v}/\vec{x}]$ evaluates to either $\mathtt{inl}\, v'$ or $\mathtt{inr}\, v'$. Let us say it evaluates to $\mathtt{inl}\, v'$. Then also, by the induction hypothesis,

$$[\![t']\!]_{[\![\vec{\tau}]\!]^p}(\vec{d}) \preccurlyeq_{\tau'+\tau''}^{(\vec{\tau},\vec{\rho})} \mathtt{inl}\, v'.$$

This means that there exists an s such that $[\![t']\!]_{[\![\vec{\tau}]\!]^p}(\vec{d}) = \mathtt{inl}\; s$ and $s \preccurlyeq_{\tau'}^{(\vec{\tau},\vec{\rho})} v'$. Since $[\![t'']\!]_{[\![\vec{\tau}]\!]^p}(\vec{d}, s) = [\![t]\!]_{[\![\vec{\tau}]\!]^p}(\vec{d}) \downarrow$, the induction hypothesis on t'' tells us that there exists a v such that $t''[\vec{\tau}/\vec{\alpha}][\vec{v}/\vec{x}, v'/x] \Downarrow v$ and

$$[\![t'']\!]_{[\![\vec{\tau}]\!]^p}(\vec{d}, s) \preccurlyeq_{\tau}^{(\vec{\tau},\vec{\rho})} v.$$

This implies $t[\vec{\tau}/\vec{\alpha}][\vec{v}/\vec{x}] \Downarrow v$ and $[\![t]\!]_{[\![\vec{\tau}]\!]^p}(\vec{d}) \preccurlyeq_{\tau}^{(\vec{\tau},\vec{\rho})} v$, as required.

$t = \langle t', t'' \rangle$.

This case is an easy analysis as above.

$t = \pi_1(t')$.

This case is an easy analysis as above.

$t = \star$.

This case is trivial.

$t = \lambda x : \sigma.\, t'$.

Suppose $[\![t]\!]_{[\![\vec{\tau}]\!]^p}(\vec{d}) \downarrow$. By Lemma 7.10, we must show that if we are also given v, d such that $d \preccurlyeq_{\sigma}^{(\vec{\tau},\vec{\rho})} v$ and $[\![t]\!]_{[\![\vec{\tau}]\!]^p}(\vec{d})(d) \downarrow$, then there exists a v' such that $t'[\vec{\tau}/\vec{\alpha}][\vec{v}/\vec{x}, v/x] \Downarrow v'$ and $[\![t]\!]_{[\![\vec{\tau}]\!]^p}(\vec{d})(d) \preccurlyeq_{\tau}^{(\vec{\tau},\vec{\rho})} v'$. But since $[\![t]\!]_{[\![\vec{\tau}]\!]^p}(\vec{d})(d) = [\![t']\!]_{[\![\vec{\tau}]\!]^p}(\vec{d}, d)$, this is just the induction hypothesis on t'.

$t = t'(t'')$.

Suppose $[\![t]\!]_{[\![\vec{\tau}]\!]^p}(\vec{d}) \downarrow$. Then $[\![t']\!]_{[\![\vec{\tau}]\!]^p}(\vec{d}) \downarrow$ and $[\![t'']\!]_{[\![\vec{\tau}]\!]^p}(\vec{d}) \downarrow$ also, so, by the induction hypothesis, there exists v, e such that $t''[\vec{\tau}/\vec{\alpha}][\vec{v}/\vec{x}] \Downarrow v$, $[\![t'']\!]_{[\![\vec{\tau}]\!]^p}(\vec{d}) \preccurlyeq_{\tau'}^{(\vec{\tau},\vec{\rho})} v$ and $t'[\vec{\tau}/\vec{\alpha}][\vec{v}/\vec{x}] \Downarrow \lambda x : \tau'.\, e$ and $[\![t']\!]_{[\![\vec{\tau}]\!]^p}(\vec{d}) \preccurlyeq_{\tau'\to\tau}^{(\vec{\tau},\vec{\rho})} \lambda x : \tau'.\, e$. This means that since

$$[\![t']\!]_{[\![\vec{\tau}]\!]^p}(\vec{d})([\![t'']\!]_{[\![\vec{\tau}]\!]^p}(\vec{d})) = [\![t]\!]_{[\![\vec{\tau}]\!]^p}(\vec{d}) \downarrow,$$

there exists a v' such that $e[v/x] \Downarrow v'$, which implies that $t[\vec{\tau}/\vec{\alpha}][\vec{v}/\vec{x}] \Downarrow v'$ and $[\![t]\!]_{[\![\vec{\tau}]\!]^p}(\vec{d}) \preccurlyeq_{\tau}^{(\vec{\tau},\vec{\rho})} v'$.

$t = \mathtt{elim}\,(t')$.

Suppose $[\![t]\!]_{[\![\vec{\tau}]\!]^p}(\vec{d}) \downarrow$. Then $[\![t']\!]_{[\![\vec{\tau}]\!]^p}(\vec{d}) \downarrow$ also, and by the induction hypothesis, there exists a v' such that $t'[\vec{\tau}/\vec{\alpha}][\vec{v}/\vec{x}] \Downarrow v'$ and $[\![t']\!]_{[\![\vec{\tau}]\!]^p}(\vec{d}) \preccurlyeq_{\mathrm{rec}\;\alpha.\sigma}^{(\vec{\tau},\vec{\rho})} v'$. Since t' is of recursive type, $v' = \mathtt{intro}\,(v)$ for some v, which means that $t[\vec{\tau}/\vec{\alpha}][\vec{v}/\vec{x}] \Downarrow v$, and, by Lemma 7.14,

$$[\![t]\!]_{[\![\vec{\tau}]\!]^p}(\vec{d}) \preccurlyeq_{\sigma[\mathrm{rec}\;\alpha.\sigma/\alpha]}^{(\vec{\tau},\vec{\rho})} v.$$

$t = \texttt{intro}\,(t')$.

Suppose $[\![t]\!]_{[\![\vec{\tau}]\!]^p}(\vec{d})\downarrow$. Then $[\![t']\!]_{[\![\vec{\tau}]\!]^p}(\vec{d})\downarrow$ also, and, by the induction hypothesis, there exists a v' such that $t'[\vec{\tau}/\vec{\alpha}][\vec{v}/\vec{x}] \Downarrow v'$ and $[\![t']\!]_{[\![\vec{\tau}]\!]^p}(\vec{d}) \preccurlyeq^{(\vec{\tau},\vec{\rho})}_{\sigma[\mathrm{rec}\ \alpha.\sigma/\alpha]} v'$. This implies $t[\vec{\tau}/\vec{\alpha}][\vec{v}/\vec{x}] \Downarrow \texttt{intro}\,v'$, and, by Lemma 7.14, $[\![t]\!]_{[\![\vec{\tau}]\!]^p}(\vec{d}) \preccurlyeq^{(\vec{\tau},\vec{\rho})}_{\mathrm{rec}\ \alpha.\sigma} \texttt{intro}\,v'$.

$t = t'(\tau)$.

In this case $\vec{\alpha} \vdash \tau$ and t' is of polymorphic type, say $t' : \prod \alpha.\,\sigma$. Suppose we have $[\![t'(\tau)]\!]_{[\![\vec{\tau}]\!]^p}(\vec{d})\downarrow$. Then $[\![t']\!]_{[\![\vec{\tau}]\!]^p}(\vec{d})\downarrow$ also, so the induction hypothesis tells us that there exists a t'' such that $t'[\vec{\tau}/\vec{\alpha}][\vec{v}/\vec{x}] \Downarrow \Lambda\alpha.\,t''$ and $[\![t']\!]_{[\![\vec{\tau}]\!]^p}(\vec{d}) \preccurlyeq^{(\vec{\tau},\vec{\rho})}_{\prod \alpha.\sigma} \Lambda\alpha.\,t''$. By Lemma 7.10, and since $[\![t']\!]_{[\![\vec{\tau}]\!]^p}(\vec{d})([\![\tau[\vec{\tau}/\vec{\alpha}]]\!]) = [\![t'(\tau)]\!]_{[\![\vec{\tau}]\!]^p}(\vec{d})\downarrow$, there exists a v such that

$$t''[\tau[\vec{\tau}/\vec{\alpha}]/\alpha] \Downarrow v$$

and

$$[\![t'(\tau)]\!]_{[\![\vec{\tau}]\!]^p}(\vec{d}) \preccurlyeq^{(\vec{\tau},\tau[\vec{\tau}/\vec{\alpha}],\vec{\rho},\preccurlyeq^{(\vec{\tau},\vec{\rho})}_{\tau})}_{\sigma} v.$$

But $t''[\tau[\vec{\tau}/\vec{\alpha}]/\alpha] \Downarrow v$ implies $t'(\tau)[\vec{\tau}/\vec{\alpha}] \Downarrow v$, and since, by Lemma 7.14,

$$\preccurlyeq^{(\vec{\tau},\tau[\vec{\tau}/\vec{\alpha}],\vec{\rho},\preccurlyeq^{(\vec{\tau},\vec{\rho})}_{\tau})}_{\sigma} \iff \preccurlyeq^{(\vec{\tau},\vec{\rho})}_{\sigma[\tau/\alpha]},$$

we have proved that there exists a v such that

$$t'(\tau)[\vec{\tau}/\vec{\alpha}] \Downarrow v$$

and

$$[\![t'(\tau)]\!]_{[\![\vec{\tau}]\!]^p}(\vec{d}) \preccurlyeq^{(\vec{\tau},\vec{\rho})}_{\sigma[\tau/\alpha]} v,$$

as required.

$t = \Lambda\alpha.\,t'$.

Since t is a value, it is clear that $[\![t]\!]_{[\![\vec{\tau}]\!]^p}(\vec{d})\downarrow$ and $t[\vec{\tau}/\vec{\alpha}][\vec{v}/\vec{x}]\Downarrow$. What we need to show is that if we are given a closed type τ' such that $[\![t]\!]_{[\![\vec{\tau}]\!]^p}(\vec{d})([\![\tau']\!]^p)\downarrow$ and an admissible relation ρ, then there exist v such that

$$t'[\vec{\tau}/\vec{\alpha}][\vec{v}/\vec{x}][\tau'/\alpha] \Downarrow v$$

and

$$[\![t]\!]_{[\![\vec{\tau}]\!]^p}(\vec{d})([\![\tau']\!]^p) \preccurlyeq^{(\vec{\tau},\tau',\vec{\rho},\rho)}_{\sigma} v.$$

But since

$$[\![t]\!]_{[\![\vec{\tau}]\!]^p}(\vec{d})([\![\tau']\!]^p) = [\![t']\!]_{[\![\vec{\tau}]\!]^p,[\![\tau']\!]^p}(\vec{d})$$

and

$$t'[\vec{\tau}/\vec{\alpha}][\vec{v}/\vec{x}][\tau'/\alpha] = t'[\vec{\tau}/\vec{\alpha},\tau'/\alpha][\vec{v}/\vec{x}],$$

this is just the induction hypothesis for t'. □

8. Reasoning using the model

The per-model of PolyFPC is parametric by construction since the interpretations of types have a built-in relational interpretation ($[\![\vec{\alpha} \vdash \sigma]\!]^r$) satisfying identity extension ($[\![\vec{\alpha} \vdash \sigma]\!]^r(eq_{\vec{R}}) = eq_{[\![\vec{\alpha}\vdash\sigma]\!]^p(\vec{R})}$). This means that the model can be used to verify parametricity arguments about PolyFPC programs, as the next example shows.

Example 8.1. This is an adaptation of a standard argument for showing data abstraction results using parametricity. We will use a type for natural numbers, which can be defined to be the recursive type $\mathtt{nat} = \mathrm{rec}\ \alpha. \alpha + 1$. Using standard arguments, one can define terms representing the usual arithmetic operations on this type. Birkedal *et al.* (2007) and Birkedal *et al.* (2006b) showed that the type nat considered as a PILL_Y type is interpreted as the admissible per $\{(\bot, \bot)\} \cup \{(\underline{n}, \underline{n}) \mid n \in \mathbb{N}\}$, for $\underline{n}$ a family of distinct incomparable elements of D. This means that in the PolyFPC model, $[\![\mathtt{nat}]\!]^p$ is the chain complete per $\{(\underline{n}, \underline{n}) \mid n \in \mathbb{N}\}$.

Suppose we are writing an FPC program P computing a natural number using an abstract data type Counter with operations New : Counter, Increment : Counter $\to$ Counter and Read : Counter $\to$ nat. We could write P as a polymorphic program of type

$$\prod \mathtt{Counter}. \mathtt{Counter} \to (\mathtt{Counter} \to \mathtt{Counter}) \to (\mathtt{Counter} \to \mathtt{nat}) \to \mathtt{nat}$$

which can be instantiated later with a concrete implementation of the type Counter. We aim to show that the result of running P on the obvious implementation of Counter as nat with New $= 0$, Increment being the successor function, and Read being the identity, is the same as running P on the intuitively equivalent implementation of Counter in which Increment adds two to the counter, and Read is division by two (rounded up to an integer). More precisely, we show that the programs

$$P\ \mathtt{nat}\ 0\ (\lambda x : \mathtt{nat}. x + 1)\ (\lambda x : \mathtt{nat}. x) \tag{11}$$
$$P\ \mathtt{nat}\ 0\ (\lambda x : \mathtt{nat}. x + 2)\ (\lambda x : \mathtt{nat}. \mathtt{Div}(x, 2)) \tag{12}$$

are contextually equivalent, which in this case means that one terminates if the other does, and if they do terminate, they return the same number. Here Div denotes an implementation of division.

Consider the relation $A : \mathrm{AdmRel}_{\mathbf{CCP}}([\![\mathtt{nat}]\!]^p, [\![\mathtt{nat}]\!]^p)$ given by the set $\{([\underline{n}], [\underline{2n}]) \mid n \in \mathbb{N}\}$. Clearly, we have $([\![0]\!], [\![0]\!]) \in A$, and the pair $([\![\lambda x : \mathtt{nat}. x + 1]\!], [\![\lambda x : \mathtt{nat}. x + 2]\!])$ maps elements related in A to elements related in LA. Similarly, the pair $([\![\lambda x : \mathtt{nat}. x]\!]$, $[\![\lambda x : \mathtt{nat}. \mathtt{Div}(x, 2)]\!])$ maps elements related in A to elements related in $L(eq_{[\![\mathtt{nat}]\!]^p})$. We know that in the model, $[\![P]\!]$ is related to itself in

$$L([\![\prod \mathtt{Counter}. \mathtt{Counter} \to (\mathtt{Counter} \to \mathtt{Counter}) \to (\mathtt{Counter} \to \mathtt{nat}) \to \mathtt{nat}]\!]^r).$$

This, together with the above observations, implies that the pair

$$([\![P\ \mathtt{nat}\ 0\ (\lambda x : \mathtt{nat}. x + 1)\ (\lambda x : \mathtt{nat}. x)]\!], [\![P\ \mathtt{nat}\ 0\ (\lambda x : \mathtt{nat}. x + 2)\ (\lambda x : \mathtt{nat}. \mathtt{Div}(x, 2))]\!])$$

is in $L[\![\mathtt{nat}]\!]^r$. Since nat is a closed type, $[\![\mathtt{nat}]\!]^r$ is simply equality on $[\![\mathtt{nat}]\!]^p$, so the programs (11) and (12) have equal denotation and thus by adequacy (Corollary 7.8) are contextually equivalent.

The use of parametricity in this model is unusual because of the mix of parametricity and partiality. Very often, when reasoning with parametricity, one instantiates the parametricity principle with graphs of functions, but since only total functions between chain complete pers by their graphs give rise to well-defined relations between the pers, when reasoning about the graph of a PolyFPC-term, we must first prove that it is total. Example 8.2 shows how the reasoning fails when we use graphs of partial functions.

Example 8.2. Consider the PolyFPC program $P = \Lambda\alpha.\, \lambda x : \alpha.\, (\lambda y : \alpha.\, *)x$, which has type $\prod \alpha.\, \alpha \to 1$. By parametricity, one would expect, for any pair of types α, β, and any relation R between them, that if $R(x, y)$, then $P(\alpha)(x) = P(\beta)(y)$. But consider for the relation R the graph of the function $\lambda x : 1.\, \Omega_1 : 1 \to 1$, where Ω_1 denotes a non-terminating program of type 1. Then $R(\star, \Omega_1)$, but $P(1)(\star)$ terminates, whereas $P(1)(\Omega_1)$ does not.

In future work it will be interesting to lift the parametricity principle of the model to a logic on PolyFPC. Corollary 7.8 should verify the logic in the sense that two terms that are provably equal in the logic should be ground contextually equivalent. Since the logic reasons about partial functions, it needs to include a termination predicate $(-)\downarrow$. The mix of parametricity and partiality will have the following consequences for the logic:

— Only total functions will have graphs that can be used to instantiate the parametricity principle.

— The relational interpretation of the $\to$ type constructor will relate f to g in $R \to S$ for relations R and S if and only if $f\downarrow \iff g\downarrow$, and, furthermore, for all $(x, y) \in R$, $f(x)\downarrow \iff g(y)\downarrow$ and $f(x)\downarrow$ implies $S(f(x), g(y))$.

— The parametricity principle in the logic will say that two terms e, f of, say, closed type $\prod \alpha.\, \sigma$, are ground contextually equivalent if and only if $e\downarrow \iff f\downarrow$ and, furthermore, for all pairs of types τ, τ' and any relation R between them, $e(\tau)\downarrow \iff f(\tau')\downarrow$ and $e(\tau)\downarrow$ implies $(e(\tau), f(\tau')) \in \sigma[R]$.

In addition to the parametricity principle, the logic should also include reasoning principles for recursive types, as in Pitts work (Pitts 1996). These are verified by the model because, as we have proved, the recursive types in the model are initial dialgebras.

Related recent work by Johann and Voigtländer (Voigtländer and Johann 2006) develops similar parametricity principles using operational methods, but in a setting without general recursive types as here.

Remark 8.1. The recursive types of PolyFPC were modelled here using encodings in PILL$_Y$. In particular, they were not encoded in PolyFPC using parametricity principles. In fact, the usual encodings of inductive and coinductive types using parametricity known from second-order lambda calculus cannot be used to encode recursive types in PolyFPC. For example, the type $\prod \alpha.\, \alpha \to \alpha$, which in second-order lambda calculus is a unit type, is interpreted in our model as the three point per

$$\{\{\bot\}, \{\langle \imath, \bot \rangle\}, \{\langle \imath, \imath \rangle\}\}.$$

Syntactically, this corresponds to the three observationally different terminating programs of this type:

$$\Lambda\alpha.\, \Omega_{\alpha\to\alpha}, \qquad \Lambda\alpha.\, \lambda x : \alpha.\, \Omega_\alpha, \qquad \Lambda\alpha.\, \lambda x : \alpha.\, x,$$

where Ω_σ denotes the non-terminating program of type σ.

9. Conclusions

By showing that the solutions to recursive domain equations in the linear part of the calculus PILL_Y can be used to interpret recursive types in languages with no linearity, we have shown that PILL_Y with parametric polymorphism is a useful axiomatic setup for domain theory. However, since PILL_Y is a type theory with polymorphism, it would be reasonable to expect that the constructed interpretation of FPC into PILL_Y could be extended to PolyFPC, and indeed this would help justify the theory as a theory for domain theory and polymorphism since this would be something that could not be done in classical domain theory. Unfortunately, I see no way of doing so in general. The problem seems to be that there is no way of quantifying over all coalgebras in PILL_Y. The polymorphic types give a way of quantifying over all types, and the type constructors give a way of talking about all maps of type $\alpha \multimap !\alpha$ for a given type α, but there is no way to quantify over all coalgebra maps on a given type in PILL_Y.

As we have seen, in the special case of the admissible per model, the interpretation of FPC does extend to full PolyFPC. This particular model is interesting for two other reasons: the resulting PolyFPC model has a particularly simple presentation in this case, and it is computationally adequate. The latter result gives a strong connection between the syntax and semantics of PolyFPC, which is similar to that known for FPC in classical domain theory, and it means that the reasoning principles for parametricity and recursive types present in the model can be lifted to a logic for PolyFPC. The parametricity principle in the model can be used to prove modularity principles for PolyFPC along the lines of Pitts (2005).

As mentioned in the introduction, other models of languages with recursive types have been suggested using pers over reflexive domains (Abadi and Plotkin 1990). The models mentioned in *loc. cit.* are based on a different category of pers, namely, admissible pers satisfying a uniformity condition among other axioms, which makes possible an adaptation of the solution of recursive domain equations from domain theory to the category of pers in question. The definition of the category of pers used in the model presented here is surprisingly simple in comparison because we use parametricity to solve recursive domain equations. But the real difference between the two models is that this paper presents a *parametric* model of PolyFPC.

Appendix A. Proof of Proposition 5.4

Recall that we use F to denote the action of F_{coalg} on morphisms of coalgebras as these functors agree on morphisms, so whenever we write $F_{\mathrm{coalg}}(\xi)$ in this proof, we actually mean F_{coalg} applied to a coalgebra ξ considered as an object in $\mathbf{C}^!$.

We first show that ξ is a coalgebra for the comonad. Note first that ϵ is a map of F-algebras in the sense that

$$\begin{tikzcd}
F(!\mu X.F(X)) \arrow[r, "F_{\mathrm{coalg}}(\delta)"] \arrow[d, "F(\epsilon)"'] & !F(!\mu X.F(X)) \arrow[r, "!F(\epsilon)"] & !F(\mu X.F(X)) \arrow[r, "!in"] & !\mu X.F(X) \arrow[d, "\epsilon"] \\
F(\mu X.F(X)) \arrow[rrr, "in"] & & & \mu X.F(X)
\end{tikzcd}$$

commutes since

$$\begin{aligned} in \circ F(\epsilon) &= in \circ F(\epsilon) \circ \epsilon \circ F_{\mathrm{coalg}}(\delta) \\ &= in \circ \epsilon \circ !F(\epsilon) \circ F_{\mathrm{coalg}}(\delta) \\ &= \epsilon \circ !in \circ !F(\epsilon) \circ F_{\mathrm{coalg}}(\delta). \end{aligned}$$

So $\epsilon \circ \xi$ is a map of algebras from *in* to itself, which means that it is the identity since *in* is the initial algebra.

To show the second equation $!\xi \circ \xi = \delta \circ \xi$, we first show that δ is a map of F-algebras in the sense that the outer diagram in

$$\begin{tikzcd} F(!\mu X.F(X)) \arrow[r, "F_{\mathrm{coalg}}(\delta)"] \arrow[d, "F(\delta)"'] & !F(!\mu X.F(X)) \arrow[r, "!F(\epsilon)"] \arrow[d, "!F(\delta)"] & !F(\mu X.F(X)) \arrow[r, "!in"] & !\mu X.F(X) \arrow[d, "\delta"] \\ F(!!\mu X.F(X)) \arrow[r, "F_{\mathrm{coalg}}(\delta)"] & !F(!!\mu X.F(X)) \arrow[rr, "!!in \circ !!F(\epsilon) \circ !F_{\mathrm{coalg}}(\delta) \circ !F(\epsilon)"] & & !!\mu X.F(X) \end{tikzcd} \tag{13}$$

commutes. The inner diagram on the left commutes since $\delta_{\mu X.F(X)}$ is a morphism of coalgebras from $\delta_{\mu X.F(X)}$ to $\delta_{!\mu X.F(X)}$, so, since F_{coalg} is a functor, $F(\delta_{\mu X.F(X)})$ is a morphism of coalgebras from $F_{\mathrm{coalg}}(\delta_{\mu X.F(X)})$ to $F_{\mathrm{coalg}}(\delta_{!\mu X.F(X)})$. The outer diagram commutes by the following computation

$$\begin{aligned} \delta \circ !in \circ !F(\epsilon) \circ F_{\mathrm{coalg}}(\delta) &= !!in \circ !!F(\epsilon) \circ \delta \circ F_{\mathrm{coalg}}(\delta) \\ &= !!in \circ !!F(\epsilon) \circ !F_{\mathrm{coalg}}(\delta) \circ F_{\mathrm{coalg}}(\delta) \\ &= !!in \circ !!F(\epsilon) \circ !F_{\mathrm{coalg}}(\delta) \circ !F(\epsilon) \circ !F(\delta) \circ F_{\mathrm{coalg}}(\delta) \\ &= !!in \circ !!F(\epsilon) \circ !F_{\mathrm{coalg}}(\delta) \circ !F(\epsilon) \circ F_{\mathrm{coalg}}(\delta) \circ F(\delta) \end{aligned}$$

(in the step going from the first to the second line we have used the fact that $F_{\mathrm{coalg}}(\delta)$ is a coalgebra for the comonad). We show that $!\xi$ is a map of F-algebras in the same way, that is, by showing that the diagram

$$\begin{tikzcd} F(!\mu X.F(X)) \arrow[r, "F_{\mathrm{coalg}}(\delta)"] \arrow[d, "F(!\xi)"'] & !F(!\mu X.F(X)) \arrow[r, "!F(\epsilon)"] \arrow[d, "!F(!\xi)"] & !F(\mu X.F(X)) \arrow[r, "!in"] \arrow[d, "!F(\xi)"] & !\mu X.F(X) \arrow[d, "!\xi"] \\ F(!!\mu X.F(X)) \arrow[r, "F_{\mathrm{coalg}}(\delta)"] & !F(!!\mu X.F(X)) \arrow[r, "!F(\epsilon)"] & !F(!\mu X.F(X)) \arrow[r, "!!in \circ !!F(\epsilon) \circ !F_{\mathrm{coalg}}(\delta)"] & !!\mu X.F(X) \end{tikzcd} \tag{14}$$

commutes. The inner diagram on the right is just ! of the defining diagram for ξ, and thus commutes. The inner diagram in the middle is commutative by ϵ being a natural transformation. Finally, the inner diagram on the left is commutative as $!\xi$ is a map of coalgebras from $\delta_{\mu X.F(X)}$ to $\delta_{!\mu X.F(X)}$ (this is just naturality of δ), so $F(!\xi)$ must be a map of coalgebras from $F_{\mathrm{coalg}}(\delta_{\mu X.F(X)})$ to $F_{\mathrm{coalg}}(\delta_{!\mu X.F(X)})$.

From (13) and (14) we see that $\delta \circ \xi$ and $!\xi \circ \xi$ are both maps of F-algebras from *in* to the same algebra, and thus, by the initiality of *in*, they must be equal. We conclude that ξ is a coalgebra for the comonad !, and thus an object of $\mathbf{C}^!$ as required.

To see that (3) commutes, note first that since ξ is a map of coalgebras from ξ to $\delta_{\mu X.F(X)}$, the diagram

$$\begin{tikzcd}
F(\mu X.F(X)) \arrow[r, "F(\xi)"] \arrow[d, "F_{\mathrm{coalg}}(\xi)"'] & F(!\mu X.F(X)) \arrow[d, "F_{\mathrm{coalg}}(\delta)"] \\
!F(\mu X.F(X)) \arrow[r, "!F(\xi)"] & !F(!\mu X.F(X))
\end{tikzcd}$$

commutes. Now,

$$\begin{aligned}
!in \circ F_{\mathrm{coalg}}(\xi) &= !in\circ!F(\epsilon)\circ!F(\xi) \circ F_{\mathrm{coalg}}(\xi) \\
&= !in\circ!F(\epsilon) \circ F_{\mathrm{coalg}}(\delta) \circ F(\xi) \\
&= \xi \circ in
\end{aligned}$$

(the last equation by the defining diagram for ξ), which proves the commutativity of (3).

Finally, we must prove the commutativity of (5) for the case where g is the unique map making the top square commute. The bottom square is just ! applied to the top square, and thus commutes. We show that the right-hand side of the cube commutes, which shows that g is a map of coalgebras from ξ to χ. From this it follows that $F(g)$ is a map of coalgebras from $F_{\mathrm{coalg}}(\xi)$ to $F_{\mathrm{coalg}}(\delta)$, that is, commutativity of the left-hand side of the cube, since F_{coalg} is a functor.

We prove equality of $\chi \circ g$ and $!g \circ \xi$ by showing that they are both F-algebra maps from *in* to

$$F(!X) \xrightarrow{F_{\mathrm{coalg}}(\delta_X)} !F(!X) \xrightarrow{!F(\epsilon)} !F(X) \xrightarrow{!f} !X\ ,$$

and then appealing to the initiality of *in*. To see that $!g \circ \xi$ is a map of algebras, consider the diagram

$$\begin{tikzcd}
F(\mu X.F(X)) \arrow[r, "F(\xi)"] \arrow[dddd, "in"'] & F(!\mu X.F(X)) \arrow[r, "F(!g)"] \arrow[d, "F_{\mathrm{coalg}}(\delta)"'] & F(!X) \arrow[d, "F_{\mathrm{coalg}}(\delta_X)"] \\
& !F(!\mu X.F(X)) \arrow[r, "!F(!g)"] \arrow[d, "!F(\epsilon)"'] & !F(!X) \arrow[d, "!F(\epsilon)"] \\
& !F(\mu X.F(X)) \arrow[r, "!F(g)"] \arrow[d, "!in"'] & !F(X) \arrow[d, "!f"] \\
& & \\
\mu X.F(X) \arrow[r, "\xi"] & !\mu X.F(X) \arrow[r, "!g"] & !X.
\end{tikzcd}$$

The outer diagram is the one we should prove commutes. The inner diagram on the left is the defining diagram for ξ, the topmost inner diagram to the right is the functor F_{coalg} applied to the coalgebra map $!g$ from $\delta_{\mu X.F(X)}$ to δ_X (it is a coalgebra map by naturality of δ), and the two other inner diagrams on the right are $!F(-)$ applied to the naturality diagram for ϵ and !, respectively, applied to the defining diagram for g.

To see that $\chi \circ g$ is a map of F-algebras, consider the diagram

$$\begin{tikzcd}
F(\mu X.\, F(X)) \arrow[r, "F(g)"] \arrow[ddd, "in"'] & F(X) \arrow[rr, "F(\chi)"] \arrow[ddd, "f"] \arrow[dr, "F_{\mathrm{coalg}}(\chi)"] & & F(!X) \arrow[d, "F_{\mathrm{coalg}}(\delta_X)"] \\
& & !F(X) \arrow[r, "!F(\chi)"] \arrow[dr, equal] & !F(!X) \arrow[d, "!F(\epsilon)"] \\
& & & !F(X) \arrow[d, "!f"] \\
\mu X.\, F(X) \arrow[r, "g"] & X \arrow[rr, "\chi"] & & !X
\end{tikzcd}$$

We need to show that the outer diagram commutes. The inner diagram on the left is the defining diagram for g. The lower inner diagram on the right is the assumption that f is a map of coalgebras (that is, diagram (4)) and the topmost diagram is just F_{coalg} applied to the map of coalgebras χ from χ to δ (which is a map of coalgebras since $\delta \circ \chi = !\chi \circ \chi$ holds because χ is assumed to be a coalgebra for the comonad).

As promised, we have shown that $\chi \circ g$ and $!g \circ \xi$ are both F-algebra maps from *in* to the same F-algebra, and thus, by the initiality of *in*, they must be equal. This concludes the proof of Proposition 5.4.

Acknowledgements

This paper contains ideas and creative input from Lars Birkedal, Eugenio Moggi, Rasmus Lerchedahl Petersen, Pino Rosolini and Alex Simpson. Suggestions from the anonymous referee also led to great improvements in the text.

References

Abadi, M. and Plotkin, G. (1990) A per model of polymorphism and recursive types. In: *5th Annual IEEE Symposium on Logic in Computer Science*, IEEE Computer Society Press 355–365.

Amadio, R. M. (1993) On the adequacy of per models. In: Borzyszkowski, A. M. and Sokolowski, S. (eds.) Mathematical Foundations of Computer Science 1993, 18th International Symposium. *Springer-Verlag Lecture Notes in Computer Science* **711** 222–231.

Barber, A. (1997) *Linear Type Theories, Semantics and Action Calculi*, Ph.D. thesis, Edinburgh University.

Benton, N., Bierman, G., de Paiva, V. and Hyland, M. (1992) Term assignment for intuitionistic linear logic. Technical Report 262, Computer Laboratory, University of Cambridge.

Benton, N. (1995) A mixed linear and non-linear logic: Proofs, terms and models (preliminary report). Technical report, University of Cambridge.

Birkedal, L. and Møgelberg, R. E. (2005) Categorical models of Abadi and Plotkin's logic for parametricity. *Mathematical Structures in Computer Science* **15** (4) 709–772.

Birkedal, L., Møgelberg, R. E. and Petersen, R. L. (2006a) Linear Abadi & Plotkin logic. *Logical Methods in Computer Science* **2**.

Birkedal, L., Møgelberg, R. E. and Petersen, R. L. (2006b) Parametric domain-theoretic models of polymorphic intuitionistic / linear lambda calculus. *Electr. Notes Theor. Comput. Sci* **155** 191–217.

Birkedal, L., Møgelberg, R. E. and Petersen, R. L. (2007) Domain theoretic models of parametric polymorphism. *Theoretical Computer Science* **388**.

Birkedal, L., Møgelberg, R. E. and Petersen, R. L. (2008) Category theoretic models of linear Abadi & Plotkin logic. *Theory and Application of Categories* **20** (7) 116–151.

Fiore, M. (1996) *Axiomatic Domain Theory in Categories of Partial Maps*, Distinguished Dissertations in Computer Science, Cambridge University Press.

Freyd, P. (1990a) Algebraically complete categories. In: Carboni, A., Pedicchio, M. C. and Rosolini, G. (eds.) Category Theory. Proceedings, Como 1990. *Springer-Verlag Lecture Notes in Computer Science* **1488** 95–104.

Freyd, P. (1990b) Recursive types reduced to inductive types. In: *Proceedings of the fifth IEEE Conference on Logic in Computer Science* 498–507.

Freyd, P. (1991) Remarks on algebraically compact categories. In: Fourman, M. P., Johnstone, P. and Pitts, A. M. (eds.) Applications of Categories in Computer Science. Proceedings of the LMS Symposium, Durham 1991. *London Mathematical Society Lecture Note Series* **177** 95–106.

Girard, J.-Y. (1987) Linear logic. *Theoretical Computer Science* **50** 1–102.

Hyland, J. (1988) A small complete category. *Annals of Pure and Applied Logic* **40** (2) 135–165.

Hyland, J., Johnstone, P. and Pitts, A. (1980) Tripos theory. *Mathematical Proceedings of the Cambridge Philosophical Society* **88**.

Jacobs, B. (1994) Semantics of weakening and contraction. *Annals of Pure and Applied Logic* **69** 73–106.

Jacobs, B. (1999) Categorical Logic and Type Theory. *Studies in Logic and the Foundations of Mathematics* **141**, Elsevier.

Johann, P. and Voigtländer, J. (2004) Free theorems in the presence of seq. In: *Proc. of 31st ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages, POPL 2004, Venice, Italy, 14–16 Jan. 2004*, ACM Press 99–110.

Johnstone, P. T. (2002) *Sketches of an elephant: a topos theory compendium. Vol. 2*, Oxford Logic Guides **44**, Oxford University Press.

Kock, A. (1970) Monads on symmetric monoidal closed categories. *Archiv der Mathematik* **21** 1–10.

Kock, A. (1972) Strong functors and monoidal monads. *Archiv der Mathematik* **23** 113–120.

Lambek, J. and Scott, P. (1986) *Introduction to higher order categorical logic*, Cambridge University Press.

Mac Lane, S. (1971) *Categories for the Working Mathematician*, Springer-Verlag.

Maneggia, P. (2004) *Models of Linear Polymorphism*, Ph.D. thesis, University of Birmingham.

Maraist, J., Odersky, M., Turner, D. N. and Wadler, P. (1999) Call-by-name, call-by-value, call-by-need and the linear lambda calculus. *Theoretical Computer Science* **228** (1-2) 175–210.

Møgelberg, R. E. (2005) *Categorical and domain theoretic models of parametric polymorphism*, Ph.D. thesis, IT University of Copenhagen.

Pitts, A. (1996) Relational properties of domains. *Information and Computation* **127** 66–90.

Pitts, A. M. (2005) Typed operational reasoning. In: Pierce, B. C. (ed.) *Advanced Topics in Types and Programming Languages*, The MIT Press 245–289.

Plotkin, G. (1985) Lectures on predomains and partial functions. Notes for a course given at the Center for the Study of Language and Information, Stanford.

Plotkin, G. (1993a) Second order type theory and recursion. Notes for a talk at the Scott Fest.

Plotkin, G. D. (1993b) Type theory and recursion (extended abstract). In: *Proceedings, Eighth Annual IEEE Symposium on Logic in Computer Science, Montreal, Canada*, IEEE Computer Society Press 374.

Plotkin, G. and Abadi, M. (1993) A logic for parametric polymorphism. In: Typed lambda calculi and applications (Utrecht 1993). *Springer-Verlag Lecture Notes in Computer Science* **664** 361–375.

Reynolds, J. (1983) Types, abstraction, and parametric polymorphism. *Information Processing* **83** 513–523.

Robinson, E. and Rosolini, G. (1994) Reflexive graphs and parametric polymorphism. In: Abramsky, S. (ed.) *Proc. 9th Symposium in Logic in Computer Science, Paris*, IEEE Computer Society Press 364–371.

Scott, D. (1970) Outline of a mathematical theory of computation. In: *4th Annual Princeton Conference on Information Sciences and Systems* 169–176.

Scott, D. (1976) Data types as lattices. *SIAM Journal of Computing* **5** (3) 522–587.

Tse, S. and Zdancewic, S. (2004) Translating dependency into parametricity. *j-SIGPLAN* **39** (9) 115–125.

van Oosten, J. (2008) Realizability; An Introduction to its Categorical Side. *Studies in Logic and the Foundations of Mathematics* **152**, Elsevier.

Voigtländer, J. and Johann, P. (2006) Selective strictness and parametricity in structural operational semantics. Technical Report TUD-FI06-02, Technische Universität Dresden.