Interpreting Polymorphic FPC into domain theoretic models of parametric polymorphism

Rasmus Ejlers Møgelberg *

DISI, Università di Genova
mogelberg@disi.unige.it

Abstract. This paper shows how parametric PILL_Y (Polymorphic Intuitionistic / Linear Lambda calculus with a fixed point combinator Y) can be used as a metalanguage for domain theory, as originally suggested by Plotkin more than a decade ago. Using recent results about solutions to recursive domain equations in parametric models of PILL_Y, we show how to interpret FPC in these. 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 the first model of a language with parametric polymorphism, recursive terms and recursive types in a non-linear setting.

1 Introduction

Parametric polymorphism is an important reasoning principle for several reasons. One is that it provides proofs of modularity principles [27] and other results based on "information hiding" such as security principles (see for example [28]). Another is that it can be used to make simple type theories surprisingly expressive by encoding inductive and coinductive types using polymorphism. If further parametric polymorphism is combined with fixed points on the term level, inductive and coinductive types coincide, and Freyd's theory of algebraically compact categories provides solutions to general type equations. However, when introducing fixed points the parametricity principle must be weakened for the theory to be consistent. Plotkin [25, 23] suggested using the calculus PILL_Y (Polymorphic Intuitionistic / Linear Lambda calculus with a fixed point combinator Y), which in combination with parametricity would have inductive, coinductive and recursive types in the linear part of the calculus. This theory was worked out in details, along with a category theoretic treatment by Birkedal, Møgelberg and Petersen [8, 6, 7, 20], see also [10]. In *loc. cit.* a concrete model of PILL_Y is constructed using "admissible" partial equivalence relations (pers) over a reflexive domain. The theory of admissible pers can be seen as a domain theory for (impredicative) polymorphism.

Plotkin suggested using parametric PILL_Y 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 — a simply typed

* This work is sponsored by Danish Research Agency stipend no. 272-05-0031

lambda calculus with recursive term definitions and 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 interpretation uses a category of coalgebras, and the resulting translation is basically an extension of Girard's interpretation of intuitionistic logic into linear logic presented in [16] and developed on term level in [19]. The new technical contribution in this part of the paper is the treatment of recursive types.

Of particular interest is the example of the model of admissible pers. When writing out the model of FPC in this case, it becomes clear that it also models parametric polymorphism. We use this to show that PolyFPC, an extension of FPC with polymorphism defined below, can be modeled soundly in the per-model. In fact this model is computationally adequate. The model is to the authors knowledge the first model of the combination of parametric polymorphism, recursive terms and recursive types in a non-linear setting. 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 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 [21], but using denotational methods. The model is also interesting because of the mix of parametricity and partiality, a combination which, as earlier research has shown, requires an alternative formulation of parametricity, such as the one suggested in [17]. This paper sketches the resulting parametricity principle derivable from the model. In future work, the parametric reasoning in the model will be lifted to a logic for parametricity for PolyFPC.

A related paper is [1], in which 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 *loc. cit.*and this paper is that the present model is parametric, and in our model the recursive types are constructed using parametricity.

The paper is organized 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, and Section 4 shows how to model FPC in general models of PILL_Y. In Section 5 the interpretation of PolyFPC in the per-model resulting from the general theory is written out in detail and computational adequacy is formulated. Unfortunately the proof of adequacy is omitted for reasons of space. Finally, Section 6 discusses reasoning about parametric polymorphism for PolyFPC using the per-model.

Acknowledgments. The paper contains ideas and creative input from the following people: Lars Birkedal, Eugenio Moggi, Rasmus Lerchedahl Petersen, Pino Rosolini and Alex Simpson.

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 [2] extended with polymorphism and fixed points for terms. This section sketches the calculus, but the reader is referred to [9, 20] for full details.

Types of PILL$_Y$ are given by the grammar

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

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

$$\begin{aligned} t ::= {} & x \mid \star \mid Y \mid \lambda^\circ x\colon \sigma.t \mid t\, t \mid t \otimes t \mid !t \mid \Lambda\alpha\colon \mathsf{Type}.\, t \mid t(\sigma) \mid \\ & \text{let } x\colon \sigma \otimes y\colon \tau \text{ be } t \text{ in } t \mid \text{let } !x\colon \sigma \text{ be } t \text{ in } t \mid \text{let } \star \text{ be } t \text{ in } t. \end{aligned}$$

Terms have two contexts of ordinary variables — a context of linear variables and a context of intuitionistic variables. We refer the reader to *loc. cit.* for the term formation rules of the calculus and the equality theory for terms. The term constructor $\lambda^\circ x\colon \sigma.t$ 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 $Y\colon \prod\alpha.\, (\alpha \to \alpha) \to \alpha$.

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 includes all important models known by the author (except some constructed from syntax), since the theory of the next sections is much simpler in this case.

Suppose $\mathbf{C}$ is a linear category, i.e., a symmetric monoidal closed category with a symmetric monoidal comonad ! satisfying a few extra properties as described for example in [18, 20]. We shall write $\sigma \multimap \tau$ for morphisms in $\mathbf{C}$. If further any functor $\mathbf{C}_0^{n+1} \to \mathbf{C}$, where $\mathbf{C}_0$ denotes the objects of $\mathbf{C}$ considered as a discrete category, has a right Kan extension along the projection $\mathbf{C}_0^{n+1} \to \mathbf{C}_0^n$, then one can form a model of PILL, the subset of PILL$_Y$ not including the fixed point combinator (for a full model of PILL$_Y$, a term modeling the fixed point combinator must exist). Types with n free type variables are modeled as functors $\mathbf{C}_0^n \to \mathbf{C}$ (or equivalently maps $\mathbf{C}_0^n \to \mathbf{C}_0$) by modeling $\boldsymbol{\alpha} \vdash \alpha_i$ as the i'th projection, $\otimes, I, \multimap$ using the symmetric monoidal structure, ! using the comonad and polymorphism using the Kan-extensions.

A category theoretic definition of what a *parametric* model of PILL$_Y$ is, is given in [20], but we shall not repeat that now. Instead we mention that the per-model described below is parametric, and we sketch the model theoretic formulations of the consequences of parametricity.

If $\mathbf{C}$ is a parametric model of PILL$_Y$, one can prove that it has, among other type constructions, products and coproducts, and that one can solve a large class of recursive type equations. Syntactically, a recursive type equation is usually given by a type 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, which in the case of α ocurring only positively in σ (e.g. if $\sigma = \alpha + 1$) means an initial algebra or final coalgebra for the functor induced by σ.

For the more general case of both positive and negative occurences of α in σ (such as $\sigma = (\alpha \to \alpha) + 1$), one can split the occurences of α in σ into positive and negative and obtain a functor of mixed variance. This way any type $\alpha_1, \dots \alpha_n \vdash \sigma$ in PILL$_Y$ induces a functor $(\mathbf{C}^{\mathrm{op}} \times \mathbf{C})^n \to \mathbf{C}$, which is strong in the sense that there exists a PILL$_Y$ term of type

$$\prod \boldsymbol{\alpha}, \boldsymbol{\alpha}', \boldsymbol{\beta}, \boldsymbol{\beta}' \colon \mathsf{Type}.\, (\boldsymbol{\alpha}' \multimap \boldsymbol{\alpha}) \to (\beta \multimap \boldsymbol{\beta}') \to \sigma(\boldsymbol{\alpha}, \boldsymbol{\beta}) \multimap \sigma(\boldsymbol{\alpha}', \boldsymbol{\beta}')$$

inducing the functor. In general, 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 any strong functor $F \colon (\mathbf{C}^{\mathrm{op}} \times \mathbf{C})^{n+1} \to \mathbf{C}$, there exists a strong functor $\mathrm{Fix}F \colon (\mathbf{C}^{\mathrm{op}} \times \mathbf{C})^n \to \mathbf{C}$ such that

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

where $\breve{\mathrm{Fix}}F \colon (\mathbf{C}^{\mathrm{op}} \times \mathbf{C})^n \to \mathbf{C}^{\mathrm{op}} \times \mathbf{C}$ is the functor that maps $(A_1, B_1, \dots A_n, B_n)$ to $(\mathrm{Fix}F(B_1, A_1, \dots B_n, A_n), \mathrm{Fix}F(A_1, B_1, \dots A_n, B_n))$. The functor $\mathrm{Fix}F$ is encoded in PILL$_Y$ using encodings due to Plotkin. The proof of this proceeds by first showing that any strong functor $\mathbf{C} \to \mathbf{C}$ has an initial algebra whose inverse is a final coalgebra. This phenomena called algebraic compactness has been studied by Freyd [14, 13, 15], who showed how to solve general recursive type equations in this setting. As a consequence of Freyd's theory, the functor $\mathrm{Fix}F$ also satisfies a universal property called the dinaturality condition generalizing at the same time the notion of initial algebra and final coalgebra. See [9, 20] for full details.

2.2 A per-model

We sketch a model of parametric PILL$_Y$. For details, see [9, 20]. 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.

Suppose D is a reflexive domain, i.e., a pointed complete partial order such that $[D \to D]$ is a retract of D. Then D has a combinatory algebra structure with application $x \cdot y$ defined by applying the function corresponding to x by the reflection to y. An *admissible per* is a partial equivalence relation R on D closed under chains and relating $\perp$ to itself. A map of admissible pers from R to S is a map of equivalence classes $f \colon D/R \to D/S$ such that there exists an element $e \in D$ *tracking* f in the sense that $f([x]_R) = [e \cdot x]_S$. This defines a category $\mathbf{AP}$ of admissible pers on D. We also define the subcategory $\mathbf{AP}_\perp$ of $\mathbf{AP}$ of morphisms with strict trackers, i.e., trackers satisfying $e \cdot \perp = \perp$. The category $\mathbf{AP}_\perp$ has products and also a symmetric monoidal closed structure with tensor product defined as a quotient of the product, and $R \multimap S$ as $\{(d, e) \mid d \cdot \perp = e \cdot \perp = \perp \wedge \forall x, y \in D.\, xRy \Rightarrow S(d \cdot x, e \cdot y)\}$. Finally, there is a symmetric monoidal comonad ! definable on $\mathbf{AP}_\perp$, the coKleisli category of which is $\mathbf{AP}$.

By an admissible proposition on a per R we shall mean a subset of the set of equivalence classes for R which itself constitutes an admissible per. An admissible relation between pers R and S is an admissible proposition on $R \times S$. Since $D/(R \times S) \cong D/S \times D/R$, we will often think of such a relation as a subset of the product of equivalence classes. We write $\mathbf{AdmRel}_{\mathbf{AP}_\perp}$ for the category of admissible relations on admissible pers, with as maps pairs of maps from $\mathbf{AP}_\perp$ mapping related elements to related elements. There is a reflexive graph of categories

$$\mathbf{AdmRel}_{\mathbf{AP}_\perp} \overrightarrow{\overleftarrow{\longrightarrow}} \mathbf{AP}_\perp \tag{1}$$

where the two maps from left to right map a relation to its domain and codomain respectively, and the last map maps a per to the identity relation on the per. The symmetric monoidal structure of $\mathbf{AP}_\perp$ can be extended to a symmetric monoidal structure on $\mathbf{AdmRel}_{\mathbf{AP}_\perp}$ commuting with the functors of (1) and likewise for the symmetric monoidal comonad.

In the parametric variant of the per-model, a type is modeled as a pair (σ^r, σ^p), where $\sigma^p\colon \mathbf{AP}_0^n \to \mathbf{AP}_0$ is a map as before and σ^r is a map taking an n-vector of admissible relations $(A_i\colon \mathrm{AdmRel}(R_i, S_i))_{i\leq n}$ and producing an admissible relation $\sigma^r(\boldsymbol{A})\colon \mathrm{AdmRel}(\sigma^p(\boldsymbol{R}), \sigma^p(\boldsymbol{S}))$, satisfying $\sigma^r(eq_{R_1}, \ldots, eq_{R_n}) = eq_{\sigma^p(\boldsymbol{R})}$. A term in the model from (σ^r, σ^p) to (τ^r, τ^p) (assumed to be two types with the same number of free variables) is a family of maps $(f_{\boldsymbol{R}}\colon D/\sigma^p(\boldsymbol{R}) \to D/\tau^p(\boldsymbol{R}))_{\boldsymbol{R}}$ with a common tracker, such that for all $(A_i\colon \mathrm{AdmRel}(R_i, S_i))_{i\leq n}$ and all x, y, if $([x],[y]) \in \sigma^r(\boldsymbol{A})$, then $(f_{\boldsymbol{R}}([x]), f_{\boldsymbol{S}}([y])) \in \tau^r(\boldsymbol{A})$. For further details, see [9, 20].

To see how the per-model is an example of the general models described in Section 2.1, notice that (1) describes an internal linear category in a presheaf category over the realizability topos for the combinatory algebra D. Interpreting the general construction in this presheaf category gives the per-model.

3 Polymorphic FPC

In this section we present the language PolyFPC, an extension of the language FPC, first defined by Plotkin [24] (see also [12]), with recursive function definitions and full impredicative polymorphism. This language can be considered a powerful intermediate language to be used in compilers. In later sections we will show how to interpret FPC into any PILL$_Y$-model of the form of Section 2.1 and how to interpret PolyFPC into the per-model sketched in Section 2.2.

Since PolyFPC is a language with polymorphism and general (nested) recursive types, types in the languages may have free type variables (as in PILL$_Y$) and 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 rec $\alpha.\, \sigma$ binds the type variable α. The grammar for terms is

$$\begin{aligned} t ::= {} & x \mid \star \mid \mathrm{inl}\ t \mid \mathrm{inr}\ t \mid \mathrm{case}\ t\ \mathrm{of}\ \mathrm{inl}\ x.\, t'\ \mathrm{of}\ \mathrm{inr}\ x.\, t'' \mid \langle t, t'\rangle \mid \pi_1(t) \mid \pi_2(t) \mid \\ & \lambda x\colon \sigma.\, t \mid t(t') \mid \mathrm{intro}\ t \mid \mathrm{elim}\ t \mid \mathrm{let\ rec}\ fx = t\ \mathrm{in}\ f\ t' \mid \Lambda\alpha.\, t \mid t(\tau). \end{aligned}$$

For reasons of space, we shall not repeat the well-known typing rules of FPC, but refer the reader to [12] for them. However, since our version of FPC also includes an explicit recursive term constructions, we mention the typing rule for that:

$$\frac{\boldsymbol{\alpha} \mid \boldsymbol{x}\colon \boldsymbol{\sigma}, f\colon \tau \to \tau', x\colon \tau \vdash t\colon \tau' \quad \boldsymbol{\alpha} \mid \boldsymbol{x}\colon \boldsymbol{\sigma} \vdash t'\colon \tau}{\boldsymbol{\alpha} \mid \boldsymbol{x}\colon \boldsymbol{\sigma} \vdash \mathrm{let\ rec}\ fx = t\ \mathrm{in}\ f\ t'\colon \tau'}$$

(bold letters denote sequents) and since PolyFPC also includes polymorphism, we mention the two typing rules for that:

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

The terms intro and elim introduce and eliminate terms of recursive types, e.g. if $t{:}\, \sigma[\text{rec}\ \alpha.\, \sigma/\alpha]$ then intro $t{:}\, \text{rec}\ \alpha.\, \sigma$.

In the following we shall use the terminology 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

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

Again we refer to [12] for the definition of $\Downarrow$ on FPC (where it is denoted $\rightsquigarrow$), and just mention the two new rules:

$$\frac{e' \Downarrow v' \qquad e[\lambda x{:}\, \sigma.\, \text{let rec}\ f x' = e\ \text{in}\ f\ x/f, v'/x'] \Downarrow v}{\text{let rec}\ f x' = e\ \text{in}\ f\ e' \Downarrow v}$$

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

4 Modeling 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. The inspiration for the general case will come from attempting to mimic the usual interpretation of FPC into domain theory (see for example [12, 22]) in the per-model presented in Section 2.2.

In domain theory, types of FPC are interpreted as complete partial orders (cpos) and terms as partial maps between them, i.e., in the Kleisli category for the lifting monad on the category $\mathbf{Cpo}$ of cpos. Neither of the categories $\mathbf{AP}_\perp$ or $\mathbf{AP}$ have the categorical properties needed for playing the role of $\mathbf{Cpo}$ in the adaption of the interpretation of FPC to admissible pers. Instead, the category $\mathbf{CCP}$ of chain closed pers on D and tracked maps between them is a good candidate. As in the category of admissible pers, we will consider as admissible propositions on a chain complete per R, subsets of D/R corresponding to chain complete pers, and an admissible relation is an admissible subset of the product. Admissible relations on chain complete pers form a category $\mathbf{AdmRel_{CCP}}$ where maps are pairs of maps mapping related elements to related elements. The next proposition shows how to recover $\mathbf{CCP}$ from $\mathbf{AP}_\perp$, and $\mathbf{AdmRel_{CCP}}$ from $\mathbf{AdmRel_{AP_\perp}}$.

Proposition 1. *The co-Eilenberg-Moore category* $\mathbf{AP}_\perp^!$ *for the lifting comonad* $!$ *on* $\mathbf{AP}_\perp$ *is equivalent to* $\mathbf{CCP}$*, and the co-Eilenberg-Moore category for* $!$ *on* $\mathbf{AdmRel_{AP_\perp}}$ *is equivalent to* $\mathbf{AdmRel_{CCP}}$.

Recall that the co-Eilenberg-Moore category for a comonad ! on a category $\mathbf{C}$ is the category whose objects are coalgebras for the monad (maps $\xi\colon \sigma \multimap !\sigma$ satisfying $\epsilon \circ \xi = id$ and $(!\xi) \circ \xi = \delta \circ \xi$, for ϵ, δ are counit and comultiplication) and whose morphisms are maps of coalgebras. Denoting by $\mathbf{C}^!$ the co-Eilenberg-Moore category, one may consider the Kleisli category for the induced monad L on $\mathbf{C}^!$, which we denote $(\mathbf{C}^!)_L$. This category is isomorphic to the category having the same objects as $\mathbf{C}^!$, but as morphisms from $\xi\colon \sigma \multimap !\sigma$ to $\chi\colon \tau \multimap !\tau$ all morphisms of $\mathbf{C}$ from σ to τ.

For the remainder of this section $\mathbf{C}$ will denote a parametric PILL$_Y$-model. Since $(\mathbf{C}^!)_L$ is a Kleisli category, it is reasonable to think of it as a category of partial maps for $\mathbf{C}^!$. The next lemma shows how these two categories satisfy some of the properties needed for interpreting FPC in categories of partial maps as in Fiore's dissertation [12].

Lemma 1. *The category $\mathbf{C}^!$ is cartesian and has finite coproducts. The category $\mathbf{C}^!$ is partially cartesian closed in the sense that for any object ξ of $\mathbf{C}^!$, the composite of the product functor and the inclusion*

$$\mathbf{C}^! \xrightarrow{\xi\times(-)} \mathbf{C}^! \longrightarrow (\mathbf{C}^!)_L$$

has a right adjoint $\xi \rightharpoonup (-)\colon (\mathbf{C}^!)_L \to \mathbf{C}^!$.

Proof. The first half is well known, the proof can be found in for example [3, Lemma 9]. For $\xi\colon \sigma \multimap !\sigma$, the functor $\xi \rightharpoonup (-)$ maps $\chi\colon \tau \multimap !\tau$ to the free coalgebra $\delta\colon !(\sigma \multimap \tau) \multimap !!(\sigma \multimap \tau)$.

FPC can be interpreted in $(\mathbf{C}^!)_L$ basically as in [12]. A type with n free variables is interpreted as a map $(\mathbf{C}^!_0)^n \to \mathbf{C}^!_0$, with α_i interpreted as the i'th projection and $\times, + \to$ using respectively product, coproduct and partial cartesian structure. Recursive terms is modeled using the fixed point combinator in $\mathbf{C}$. What is different from Fiore's interpretation however, is that here recursive domain equations are solved using parametricity. The next definition defines the class of domain equations which can be solved in $\mathbf{C}^!$.

Definition 1. *A functor $\sigma_{coalg}\colon ((\mathbf{C}^!)^{\mathrm{op}} \times \mathbf{C}^!)^n \to \mathbf{C}^!$ is* induced by a type*, if there exists a strong functor $\sigma\colon ((\mathbf{C})^{\mathrm{op}} \times \mathbf{C})^n \to \mathbf{C}$ making the diagram*

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

commute, where the vertical functors are the obvious forgetful functors. We say that σ induces σ_{coalg}.

If σ induces σ_{coalg} as in Definition 1 above, then σ_{coalg} extends to a functor $(((\mathbf{C}^!)_L)^{\mathrm{op}} \times (C^!)_L)^n \to (C^!)_L$, whose action on morphisms is given by σ.

We show that all recursive domain equations on $(C^!)_L$ corresponding to functors induced by types as in Definition 1 can be solved. The precise formulation of this result is Theorem 1 below. The proof proceeds by first showing that $(C^!)_L$ is algebraically compact as in the next lemma, and then applying Freyd's solution to recursive domain equations in such categories.

Lemma 2. *If the functor $\sigma_{coalg}\colon \mathbf{C}^! \to \mathbf{C}^!$ is induced by a type σ, then it has an initial algebra. Including the initial algebra into $(\mathbf{C}^!)_L$ gives an initial algebra for the functor $(\mathbf{C}^!)_L \to (\mathbf{C}^!)_L$ induced by (σ, σ_{coalg}), and the inverse of the algebra is a final coalgebra.*

Proof. We just sketch the construction of the initial algebra. As a consequence of parametricity, σ has an initial algebra $in\colon \sigma(\mu\alpha.\,\sigma) \multimap \mu\alpha.\,\sigma$ whose inverse is a final coalgebra. The object $\mu\alpha.\,\sigma$ of $\mathbf{C}$ has a coalgebra structure for ! defined as the unique map ξ making the diagram

$$\begin{tikzcd}
\sigma(\mu\alpha.\,\sigma) \arrow[rrr, "\cong", -o] \arrow[d, "\sigma(\xi)"', -o] & & & \mu\alpha.\,\sigma \arrow[d, "\xi", -o] \\
\sigma(!\mu\alpha.\,\sigma) \arrow[r, "\sigma_{\mathrm{coalg}}(\delta)", -o] & !\sigma(!\mu\alpha.\,\sigma) \arrow[r, "!\sigma(\epsilon)", -o] & !\sigma(\mu\alpha.\,\sigma) \arrow[r, "\cong", -o] & !\mu\alpha.\,\sigma
\end{tikzcd}$$

commute.

Theorem 1. *For any functor $\sigma_{coalg}\colon (\mathbf{C}^{!\mathrm{op}} \times \mathbf{C}^!)^{n+1} \to \mathbf{C}^!$ induced by a type, say σ, there exists a functor $Fix\ \sigma_{coalg}\colon ((\mathbf{C}^!)^{\mathrm{op}} \times \mathbf{C}^!)^n \to \mathbf{C}^!$ induced by a type $Fix\ \sigma$ such that*

$$\sigma_{coalg} \circ \langle id_{((\mathbf{C}^!)^{\mathrm{op}} \times \mathbf{C}^!)^n}, Fix\ \breve{\sigma}_{coalg} \rangle \cong Fix\ \sigma_{coalg} \tag{2}$$

(where the notation $(\breve{-})$ is used as in Section 2.1). The corresponding functors on $(\mathbf{C}^!)_L$ also satisfy (2) and the dinaturality condition. Finally, there exists a general construction of $Fix\ \sigma_{coalg}$ such that if $\tau_{coalg}\colon (\mathbf{C}^{!\mathrm{op}} \times \mathbf{C}^!)^m \to (\mathbf{C}^{!\mathrm{op}} \times \mathbf{C}^!)^n$ is any functor induced by a type, then

$$Fix(\sigma_{coalg} \circ (\tau_{coalg} \times id_{((\mathbf{C}^!)^{\mathrm{op}} \times \mathbf{C}^!)})) = (Fix\ \sigma_{coalg}) \circ \tau_{coalg}$$

Proof (Sketch). The recursive types are constructed as in Freyd's solution to recursive type equations as

$$\begin{aligned}
\omega(\alpha_1, \beta_1, \ldots, \alpha_n, \beta_n, \alpha) &= \mu\beta.\,\sigma(\alpha_1, \beta_1, \ldots, \alpha_n, \beta_n, \alpha, \beta) \\
\tau(\alpha_1, \beta_1, \ldots, \alpha_n, \beta_n) &= \mu\alpha.\,\sigma(\beta_1, \alpha_1, \ldots, \beta_n, \alpha_n, \omega(\alpha_1, \beta_1, \ldots, \alpha_n, \beta_n, \alpha), \alpha) \\
\mathrm{rec}\ \alpha.\,\sigma(\alpha_1, \beta_1, \ldots, \alpha_n, \beta_n, \alpha, \alpha) &= \omega(\alpha_1, \beta_1, \ldots, \alpha_n, \beta_n, \tau(\alpha_1, \beta_1, \ldots, \alpha_n, \beta_n)).
\end{aligned}$$

The dinaturality condition in $(\mathbf{C}^!)_L$ follows from the one in $\mathbf{C}$ since these have the same maps, and the last statement is an easy consequence of the construction.

Of course, to be able to use Theorem 1 for modeling recursive types, one must show that any FPC type $\alpha_1, \ldots, \alpha_n \vdash \sigma$ induces a functor $((\mathbf{C}^!)^{\mathrm{op}} \times \mathbf{C}^!)^n \to \mathbf{C}^!$ induced by a type, by splitting occurrences of free type variables into positive and negative occurrences. This is an easy induction on the structure of σ, and the case of recursive types is simply that $\mathrm{Fix}\ \sigma_{\mathrm{coalg}}$ of Theorem 1 is induced by a type for all σ_{coalg}.

Theorem 2. *FPC can be modeled soundly in* $(\mathbf{C}^!)_L$.

5 Polymorphic FPC in the per-model

The abstract analysis of Section 4 shows that our main example — the per-model — models recursive types. It also models polymorphism, and Figure 1 shows the interpretation of PolyFPC in the per-model, except the interpretation of recursive types, for which the categorical properties (dinaturality) are more useful than the concrete description as shown for instance in [22].

Types in the per-model of PolyFPC are modeled as pairs $(\llbracket \boldsymbol{\alpha} \vdash \sigma \rrbracket^p, \llbracket \boldsymbol{\alpha} \vdash \sigma \rrbracket^r)$, where $\llbracket \boldsymbol{\alpha} \vdash \sigma \rrbracket^p$ is a map $\mathbf{CCP}^n \to \mathbf{CCP}$ and $\llbracket \boldsymbol{\alpha} \vdash \sigma \rrbracket^r$ is a map taking an n-vector of admissible relations $(A_i \colon \mathbf{AdmRel}(R_i, S_i))$ (admissible in the sense of objects of $\mathbf{AdmRel_{CCP}}$) on objects of $\mathbf{CCP}$ and produces an admissible relation

$$\llbracket \boldsymbol{\alpha} \vdash \sigma \rrbracket^r(\boldsymbol{A}) \colon \mathbf{AdmRel}(\llbracket \boldsymbol{\alpha} \vdash \sigma \rrbracket^p(\boldsymbol{R}), \llbracket \boldsymbol{\alpha} \vdash \sigma \rrbracket^p(\boldsymbol{S}))$$

satisfying $\llbracket \boldsymbol{\alpha} \vdash \sigma \rrbracket^r(eq_{\boldsymbol{R}}) = eq_{\llbracket \boldsymbol{\alpha} \vdash \sigma \rrbracket^p(\boldsymbol{R})}$. In Figure 1 the symbols $1, 2$ denote two incomparable elements of D, and $\langle \cdot, \cdot \rangle$ denotes the pairing function $D \times D \to D$ definable using the combinatory algebra structure on D. The monad induced by the comonad on $\mathbf{AP}_\perp$ and $\mathbf{AdmRel}_{\mathbf{AP}_\perp}$ is denoted L. Explicitly, this monad maps a chain complete per R to $\{(\perp, \perp)\} \cup \{(\langle \iota, x \rangle, \langle \iota, y \rangle) \mid R(x, y)\}$ where ι denotes a code for the identity function on D, and an admissible relation A on chain complete pers R, S is mapped to the relation that on LR, LS that relates $[\perp]$ to $[\perp]$ and $[\langle \iota, x \rangle]$ to $[\langle \iota, y \rangle]$ if $A([x], [y])$.

$$\begin{aligned}
\llbracket \boldsymbol{\alpha} \vdash \alpha_i \rrbracket^p(\boldsymbol{R}) &= R_i \\
\llbracket \boldsymbol{\alpha} \vdash \sigma \times \tau \rrbracket^p(\boldsymbol{R}) &= \{(\langle x, y \rangle, \langle x', y' \rangle \mid \llbracket \boldsymbol{\alpha} \vdash \sigma \rrbracket^p(\boldsymbol{R})(x, x') \wedge \llbracket \boldsymbol{\alpha} \vdash \tau \rrbracket^p(\boldsymbol{R})(y, y')\} \\
\llbracket \boldsymbol{\alpha} \vdash \sigma + \tau \rrbracket^p(\boldsymbol{R}) &= \{(\langle 1, x \rangle, \langle 1, x' \rangle) \mid \llbracket \boldsymbol{\alpha} \vdash \sigma \rrbracket^p(\boldsymbol{R})(x, x')\} \cup \\
&\quad\ \{(\langle 2, y \rangle, \langle 2, y' \rangle) \mid \llbracket \boldsymbol{\alpha} \vdash \tau \rrbracket^p(\boldsymbol{R})(y, y')\} \\
\llbracket \boldsymbol{\alpha} \vdash \sigma \to \tau \rrbracket^p(\boldsymbol{R}) &= \{(e, f) \mid \forall x, y \in D.\, \llbracket \boldsymbol{\alpha} \vdash \sigma \rrbracket^p(\boldsymbol{R})(x, y) \Rightarrow L\llbracket \boldsymbol{\alpha} \vdash \tau \rrbracket^p(\boldsymbol{R})(e \cdot x, f \cdot y)\} \\
\llbracket \boldsymbol{\alpha} \vdash 1 \rrbracket^p(\boldsymbol{R}) &= \{(\perp, \perp)\} \\
\llbracket \boldsymbol{\alpha} \vdash \textstyle\prod \alpha.\, \sigma \rrbracket^p(\boldsymbol{R}) &= \{(x, y) \mid \forall S \colon \mathbf{CCP}_0.\, L\llbracket \boldsymbol{\alpha}, \alpha \vdash \sigma \rrbracket^p(\boldsymbol{R}, S)(x, y) \wedge \\
&\qquad \forall S, S' \colon \mathbf{CCP}_0.\, \forall A \colon \mathrm{AdmRel}(S, S').\, L\llbracket \boldsymbol{\alpha}, \alpha \vdash \sigma \rrbracket^r(eq_{\boldsymbol{R}}, A)([x], [y])\}
\end{aligned}$$

$$\begin{aligned}
\llbracket \boldsymbol{\alpha} \vdash \alpha_i \rrbracket^r(\boldsymbol{A}) &= A_i \\
\llbracket \boldsymbol{\alpha} \vdash \sigma \times \tau \rrbracket^r(\boldsymbol{A}) &= \{([\langle x, y \rangle], [\langle x', y' \rangle]) \mid \llbracket \boldsymbol{\alpha} \vdash \sigma \rrbracket^r(\boldsymbol{A})([x], [x']) \wedge \llbracket \boldsymbol{\alpha} \vdash \tau \rrbracket^r(\boldsymbol{A})([y], [y'])\} \\
\llbracket \boldsymbol{\alpha} \vdash \sigma + \tau \rrbracket^r(\boldsymbol{A}) &= \{([\langle 1, x \rangle], [\langle 1, x' \rangle]) \mid \llbracket \boldsymbol{\alpha} \vdash \sigma \rrbracket^r(\boldsymbol{A})([x], [x'])\} \cup \\
&\quad\ \{([\langle 2, y \rangle], [\langle 2, y' \rangle]) \mid \llbracket \boldsymbol{\alpha} \vdash \tau \rrbracket^r(\boldsymbol{A})([y], [y'])\} \\
\llbracket \boldsymbol{\alpha} \vdash \sigma \to \tau \rrbracket^r(\boldsymbol{A}) &= \{([e], [f]) \mid \forall ([x], [y]) \in \llbracket \boldsymbol{\alpha} \vdash \sigma \rrbracket^r(\boldsymbol{A}).\, ([e \cdot x], [f \cdot y]) \in L\llbracket \boldsymbol{\alpha} \vdash \tau \rrbracket^r(\boldsymbol{A})\} \\
\llbracket \boldsymbol{\alpha} \vdash 1 \rrbracket^r(\boldsymbol{A}) &= \{([\perp], [\perp])\} \\
\llbracket \boldsymbol{\alpha} \vdash \textstyle\prod \alpha.\, \sigma \rrbracket^r(\boldsymbol{A}) &= \{([x], [y]) \mid \forall S, S' \colon \mathbf{CCP}_0.\, \forall A \colon \mathrm{AdmRel}(S, S').\, L\llbracket \boldsymbol{\alpha}, \alpha \vdash \sigma \rrbracket^r(\boldsymbol{A}, A)([x], [y])\}
\end{aligned}$$

Fig. 1. Interpretation of PolyFPC in per-model

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

$$(\llbracket t \rrbracket_{\boldsymbol{R}}{:} \prod_i \llbracket \boldsymbol{\alpha} \vdash \sigma_i \rrbracket^p(\boldsymbol{R}) \to L\llbracket \boldsymbol{\alpha} \vdash \tau \rrbracket^p(\boldsymbol{R}))_{\boldsymbol{R}}$$

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

$$(\llbracket t \rrbracket_{\boldsymbol{R}}([x_1], \ldots, [x_m]), \llbracket t \rrbracket_{\boldsymbol{S}}([y_1], \ldots, [y_m])) \in L\llbracket \boldsymbol{\alpha} \vdash \tau \rrbracket^r(\boldsymbol{A}).$$

Theorem 3. *The interpretation of PolyFPC types defined in Figure 1 extends to a sound interpretation of PolyFPC.*

5.1 Computational adequacy

A τ - σ context of PolyFPC for types σ, τ, where τ is closed, is an expression C containing a place holder $-_\tau$ 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: There exists a v such that $t \Downarrow v$.

Theorem 4 (Adequacy). *For any program t of PolyFPC,* $\llbracket t \rrbracket \neq [\bot]$ *iff* $t \Downarrow$.

From Theorem 4 the following corollary giving a tight connection between the operational and denotational semantics is easily provable.

Corollary 1. *Suppose* t, t' *are two PolyFPC terms of the same type. If* $\llbracket t \rrbracket = \llbracket t' \rrbracket$ *then* $t \equiv t'$.

6 Reasoning using the model

The per-model of PolyFPC is parametric by construction, since the interpretations of types have a build-in relational interpretation ($\llbracket \boldsymbol{\alpha} \vdash \sigma \rrbracket^r$) satisfying identity extension. This means that the model can be used to verify parametricity arguments about PolyFPC programs. For example, for the usual data abstraction arguments as in [27, 21] proving that two implementations of a data type gives the same final program, one can prove using parametricity of the model, that the two programs denotations are equal, and then use Corollary 1 to prove that the programs are ground contextually equivalent.

In future work, it will be interesting to lift the parametricity of the model to a logic on PolyFPC. Corollary 1 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 proposition $(-)\downarrow$. The mix of parametricity and partiality will have the following consequences on 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 iff $f \downarrow \iff g \downarrow$, and further 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 iff $e \downarrow \iff f \downarrow$ and further, 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 $(e(\tau), f(\tau')) \in \sigma[R]$.

Related results can be found in [17], and the interpretation of $\to$ above is a symmetric version of the one in *loc. cit.*.

7 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 parametric PILL_Y is a useful axiomatic setup for domain theory. The parametric model of PolyFPC constructed by applying the general theory to the case of admissible pers can be used to reason about parametricity for PolyFPC and for example give proofs of modularity properties along the lines of [21], but this time using the denotational semantics.

In recent work, Birkedal, Møgelberg, Petersen and Varming [11] have shown how the programming language lily of Bierman, Pitts and Russo [4] gives rise to a parametric model of PILL_Y. Using this result, the techniques developed here should show how FPC can be translated into lily, but it would be interesting to see if full PolyFPC can be translated into it.

References

1. M. Abadi and G.D. Plotkin. A per model of polymorphism and recursive types. In *5th Annual IEEE Symposium on Logic in Computer Science*, pages 355–365. IEEE Computer Society Press, 1990.
2. A. Barber. *Linear Type Theories, Semantics and Action Calculi*. PhD thesis, Edinburgh University, 1997.
3. P.N. Benton. A mixed linear and non-linear logic: Proofs, terms and models (preliminary report). Technical report, University of Cambridge, 1995.
4. G. M. Bierman, A. M. Pitts, and C. V. Russo. Operational properties of Lily, a polymorphic linear lambda calculus with recursion. In *Fourth International Workshop on Higher Order Operational Techniques in Semantics, Montréal*, volume 41 of *Electronic Notes in Theoretical Computer Science*. Elsevier, September 2000.
5. L. Birkedal and R. E. Møgelberg. Categorical models of Abadi-Plotkin's logic for parametricity. *Mathematical Structures in Computer Science*, 15(4):709–772, 2005.
6. L. Birkedal, R. E. Møgelberg, and R. L. Petersen. Category theoretic models of linear Abadi & Plotkin logic. Submitted.
7. L. Birkedal, R. E. Møgelberg, and R. L. Petersen. Domain theoretic models of parametric polymorphism. Submitted.

8. L. Birkedal, R. E. Møgelberg, and R. L. Petersen. Linear Abadi & Plotkin logic. Submitted.
9. L. Birkedal, R. E. Møgelberg, and R. L. Petersen. Parametric domain-theoretic models of linear Abadi & Plotkin logic. 2005. Submitted.
10. L. Birkedal, R. E. Møgelberg, and R. L. Petersen. Parametric domain-theoretic models of polymorphic intuitionistic / linear lambda calculus. In *Proceedings of the Twenty-first Conference on the Mathematical Foundations of Programming Semantics*, 2005. To appear.
11. L. Birkedal, R.L. Petersen, R.E. Møgelberg, and C. Varming. Operational semantics and models of linear Abadi-Plotkin logic. Manuscript.
12. M. Fiore. *Axiomatic Domain Theory in Categories of Partial Maps*. Distinguished Dissertations in Computer Science. Cambridge University Press, 1996.
13. P.J. Freyd. Algebraically complete categories. In A. Carboni, M. C. Pedicchio, and G. Rosolini, editors, *Category Theory. Proceedings, Como 1990*, volume 1488 of *Lecture Notes in Mathematics*, pages 95–104. Springer-Verlag, 1990.
14. P.J. Freyd. Recursive types reduced to inductive types. In *Proceedings of the fifth IEEE Conference on Logic in Computer Science*, pages 498–507, 1990.
15. P.J. Freyd. Remarks on algebraically compact categories. In M. P. Fourman, P.T. Johnstone, and A. M. Pitts, editors, *Applications of Categories in Computer Science. Proceedings of the LMS Symposium, Durham 1991*, volume 177 of *London Mathematical Society Lecture Note Series*, pages 95–106. Cambridge University Press, 1991.
16. Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.
17. Patricia Johann and Janis Voigtländer. 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*, pages 99–110. ACM Press, New York, 2004.
18. Paola Maneggia. *Models of Linear Polymorphism*. PhD thesis, University of Birmingham, Feb. 2004.
19. Maraist, Odersky, Turner, and Wadler. Call-by-name, call-by-value, call-by-need and the linear lambda calculus. *TCS: Theoretical Computer Science*, 228:175–210, 1999.
20. R. E. Møgelberg. *Categorical and domain theoretic models of parametric polymorphism*. PhD thesis, IT University of Copenhagen, 2005.
21. A. M. Pitts. Typed operational reasoning. In B. C. Pierce, editor, *Advanced Topics in Types and Programming Languages*, chapter 7, pages 245–289. The MIT Press, 2005.
22. A.M. Pitts. Relational properties of domains. *Information and Computation*, 127:66–90, 1996.
23. G. D. Plotkin. Type theory and recursion (extended abstract). In *Proceedings, Eighth Annual IEEE Symposium on Logic in Computer Science*, page 374, Montreal, Canada, 19–23 June 1993. IEEE Computer Society Press.
24. G.D. Plotkin. Lectures on predomains and partial functions. Notes for a course given at the Center for the Study of Language and Information, Stanford, 1985.
25. G.D. Plotkin. Second order type theory and recursion. Notes for a talk at the Scott Fest, February 1993.
26. Gordon Plotkin and Martín Abadi. A logic for parametric polymorphism. In *Typed lambda calculi and applications (Utrecht, 1993)*, volume 664 of *Lecture Notes in Comput. Sci.*, pages 361–375. Springer, Berlin, 1993.
27. J.C. Reynolds. Types, abstraction, and parametric polymorphism. *Information Processing*, 83:513–523, 1983.
28. Stephen Tse and Steve Zdancewic. Translating dependency into parametricity. *j-SIGPLAN*, 39(9):115–125, September 2004.