A logic for parametric polymorphism with effects

Simpson, A., & Mogelberg, R. (2008). A logic for parametric polymorphism with effects. In Types for Proofs and Programs . (pp. 142-156). (Lecture Notes in Computer Science). Springer Berlin / Heidelberg. doi: 10.1007/978-3-540-68103-8_10

A logic for parametric polymorphism with effects

Rasmus Ejlers Møgelberg and Alex Simpson

LFCS, School of Informatics,
University of Edinburgh

Abstract. We present a logic for reasoning about parametric polymorphism in combination with arbitrary computational effects (nondeterminism, exceptions, continuations, side-effects etc.). As examples of reasoning in the logic, we show how to verify correctness of polymorphic type encodings in the presence of effects.

1 Introduction

Strachey [11] defined a polymorphic program to be *parametric* if it applies the same uniform algorithm across all of its type instantiations. Parametric polymorphism has proved to be a very useful programming language feature. However, the informal definition of Strachey does not lend itself to providing methods of verifying properties of polymorphic programs. Reynolds [10] addressed this by formulating the mathematical notion of *relational parametricity*, in which the uniformity in Strachey's definition is captured by requiring programs to preserve certain relations induced by the type structure. In the context of pure functional polymorphic languages, such as the second-order lambda-calculus, relational parametricity has proven to be a powerful principle for establishing abstraction properties, proving equivalence of programs and inferring useful properties of programs from their types alone [12].

Obtaining a useful and indeed consistent formulation of relational parametricity becomes trickier in the presence of computational effects (nondeterminism, exceptions, side-effects, continuations, etc.). Even the addition of recursion (and hence possible nontermination) to the second-order lambda-calculus causes difficulties. For this special case, Plotkin proposed second-order intuitionistic-linear type theory as a suitable framework for formulating relational parametricity [9]. This framework has since been developed by the first author and colleagues [2], but it does not adapt to general effects.

Recently, the authors have developed a more general framework that is appropriate for modelling parametric polymorphism in combination with arbitrary computational effects [4]. The framework is based on a custom-built type theory PE for combining polymorphism and effects, which is strongly influenced by Moggi's computational metalanguage [6], and Levy's call-by-push-value calculus [3]. As presented in [4], the type theory is interpreted in relationally-parametric models developed within the context of an intuitionistic set theory as the mathematical meta-theory. While this approach provides an efficient

framework for building models, the underlying principles for reasoning about the combination of parametricity and effects are left buried amongst the (considerable) semantic details. The purpose of the present article is to extract the logic for parametricity with effects that is implicit within these models, and to give a self-contained presentation of it. In particular, no understanding of the semantic setting of [4] is required.

The logic we present, builds on Plotkin and Abadi's logic for parametric polymorphism in second-order lambda-calculus [7], and is influenced by the existing refinements of this logic to linear type theory and recursion [9, 2]. The logic is built over the type theory PE, presented by the authors in [4]. As in Levy's call-by-push-value (CBPV) calculus [3], the calculus PE has two kinds of types: value types (whose elements are static values) and computation types (whose elements are dynamic effect-producing computations. The type theory allows for polymorphic quantification over value types as well as over computation types. A central result in [4] is that the algebraic operations that cause effects (as in [8]) can be given polymorphic types and satisfy a parametricity principle. For example, in a type theory for polymorphism and nondeterminism, the nondeterministic choice operation has polymorphic type $\forall \underline{X}.\, \underline{X} \to \underline{X} \to \underline{X}$, where $\underline{X}$ ranges over all computation types.

An essential ingredient in the logic we present is the division of relations into value relations and computation relations. The latter generalise the notion of admissible relations that arise in the theory of parametricity and recursion [2]. To see why such a notion is necessary for the formulation of a consistent theory of parametricity, consider the type $\forall \underline{X}.\, \underline{X} \to \underline{X} \to \underline{X}$ of a binary nondeterministic choice operation, as above. Relational parametricity, states that for all computation types $\underline{X}, \underline{Y}$ and all relations R between them, any operation of the above type must preserve R. If R were to range over arbitrary relations, then only the first and second projections would satisfy this condition, and so algebraic operations (such as nondeterministic choice) would not count as parametric. This is why a restricted class of computation relations is needed. Such relations can be thought of as relations that respect the computational structure.

This paper makes two main contributions. The first is the formulation of the logic itself, which is given in Section 3. Here, our goal is to present the logic in an intelligible way, and we omit the (straightforward) proofs of the basic properties of the logic. Our second contribution is to use the logic to formalize correctness arguments for the type theory PE. In particular, we verify that our logic for parametricity with effects proves desired universal properties for several polymorphically-defined type contructors, including existential and coinductive computation types. For this, we include as much detail as space permits.

2 A type theory for polymorphism and effects

This section recalls the type theory PE for polymorphism and effects as defined and motivated in [4]; see also [5] for an application. As mentioned in the introduction, like CBPV [3], PE has two collections of types: value types and

$$\frac{}{\Gamma, x\colon \mathsf{B} \mid - \vdash x\colon \mathsf{B}} \qquad \frac{\Gamma, x\colon \mathsf{B} \mid \Delta \vdash t\colon \mathsf{C}}{\Gamma \mid \Delta \vdash \lambda x\colon \mathsf{B}.\, t\colon \mathsf{B} \to \mathsf{C}} \qquad \frac{\Gamma \mid \Delta \vdash s\colon \mathsf{B} \to \mathsf{C} \quad \Gamma \mid - \vdash t\colon \mathsf{B}}{\Gamma \mid \Delta \vdash s(t)\colon \mathsf{C}}$$

$$\frac{\Gamma \mid \Delta \vdash t\colon \mathsf{B}}{\Gamma \mid \Delta \vdash \Lambda X.\, t\colon \forall X.\, \mathsf{B}}\ X \notin \mathrm{FTV}(\Gamma, \Delta) \qquad \frac{\Gamma \mid \Delta \vdash t\colon \forall X.\, \mathsf{B}}{\Gamma \mid \Delta \vdash t(\mathsf{A})\colon \mathsf{B}[\mathsf{A}/X]}$$

$$\frac{}{\Gamma \mid x\colon \underline{\mathsf{A}} \vdash x\colon \underline{\mathsf{A}}} \qquad \frac{\Gamma \mid x\colon \underline{\mathsf{A}} \vdash t\colon \underline{\mathsf{B}}}{\Gamma \mid - \vdash \lambda^{\circ} x\colon \underline{\mathsf{A}}.\, t\colon \underline{\mathsf{A}} \multimap \underline{\mathsf{B}}} \qquad \frac{\Gamma \mid - \vdash s\colon \underline{\mathsf{A}} \multimap \underline{\mathsf{B}} \quad \Gamma \mid \Delta \vdash t\colon \underline{\mathsf{A}}}{\Gamma \mid \Delta \vdash s(t)\colon \underline{\mathsf{B}}}$$

$$\frac{\Gamma \mid \Delta \vdash t\colon \mathsf{B}}{\Gamma \mid \Delta \vdash \Lambda \underline{X}.\, t\colon \forall \underline{X}.\, \mathsf{B}}\ \underline{X} \notin \mathrm{FTV}(\Gamma, \Delta) \qquad \frac{\Gamma \mid \Delta \vdash t\colon \forall \underline{X}.\, \mathsf{B}}{\Gamma \mid \Delta \vdash t(\underline{\mathsf{A}})\colon \mathsf{B}[\underline{\mathsf{A}}/\underline{X}]}$$

Fig. 1. Typing rules for PE.

computation types. We follow Levy's convention of distinguishing syntactically between the two by underlining computation types as in $\underline{\mathsf{A}}, \underline{\mathsf{B}}, \underline{\mathsf{C}} \ldots$. The calculus PE has polymorphic quantification over both value types and computation types, with type variables denoted $X, Y \ldots$ and $\underline{X}, \underline{Y} \ldots$ respectively. Value types and computation types are defined by the grammar

$$\begin{aligned} \mathsf{A}, \mathsf{B} &::= X \mid \mathsf{A} \to \mathsf{B} \mid \forall X.\, \mathsf{A} \mid \underline{X} \mid \forall \underline{X}.\, \mathsf{A} \mid \underline{\mathsf{A}} \multimap \underline{\mathsf{B}} \\ \underline{\mathsf{A}}, \underline{\mathsf{B}} &::= \mathsf{A} \to \underline{\mathsf{B}} \mid \forall X.\, \underline{\mathsf{A}} \mid \underline{X} \mid \forall \underline{X}.\, \underline{\mathsf{A}}\ . \end{aligned}$$

Note that the computation types form a subcollection of the value types. One semantic intuition is that value types are sets and computation types are algebras for some computational monad in the sense of Moggi [6]. In such a model, $\multimap$ is modelled by the collection of algebra homomorphisms, a set which does not in general carry a natural algebra structure and is thus a value type in PE, and the inclusion of computation types into value types is modelled by the forgetful functor mapping an algebra to its carrier. We refer the interested reader to [4] for a discussion of such models in detail.

Typing judgements of PE are of the form $\Gamma \mid \Delta \vdash t\colon \mathsf{A}$ where Γ is an ordinary context of variables, and Δ is a second context called the *stoup* subject to the following conditions: either Δ is empty or it is of the form $\Delta = z\colon \underline{\mathsf{B}}$, in which case A is also required to be a computation type. The semantic intuition for the second case is that t denotes an algebra homomorphism from $\underline{\mathsf{B}}$ to A.

The typing rules are presented in Figure 1. In them, $\Gamma \mid - \vdash t\colon \underline{\mathsf{A}}$ denotes a judgement with empty stoup, and the operation FTV returns the set of free type variables, which is defined in the obvious way. Note the following consequence of the typing rules: if $\Gamma \mid z\colon \underline{\mathsf{A}} \vdash t\colon \underline{\mathsf{B}}$ is well typed, then so is $\Gamma, z\colon \underline{\mathsf{A}} \mid - \vdash t\colon \underline{\mathsf{B}}$. Terms of PE are identified up to α-equivalence as usual.

Although the calculus PE has just a few primitive type constructors, a wide range of derived type constructors, on both value types and computation types, can be encoded using polymorphism.

$$
\begin{array}{rll}
1 & =_{\mathrm{def}} \forall X.\, X \to X & \\
\mathsf{A} \times \mathsf{B} & =_{\mathrm{def}} \forall X.\, (\mathsf{A} \to \mathsf{B} \to X) \to X & (X \notin \mathrm{FTV}(\mathsf{A},\mathsf{B})) \\
0 & =_{\mathrm{def}} \forall X.\, X & \\
\mathsf{A} + \mathsf{B} & =_{\mathrm{def}} \forall X.\, (\mathsf{A} \to X) \to (\mathsf{B} \to X) \to X & (X \notin \mathrm{FTV}(\mathsf{A},\mathsf{B})) \\
\exists X.\, \mathsf{B} & =_{\mathrm{def}} \forall Y.\, (\forall X.\, (\mathsf{B} \to Y)) \to Y & (Y \notin \mathrm{FTV}(\mathsf{B})) \\
\exists \underline{X}.\, \mathsf{B} & =_{\mathrm{def}} \forall Y.\, (\forall \underline{X}.\, (\mathsf{B} \to Y)) \to Y & (Y \notin \mathrm{FTV}(\mathsf{B})) \\
\mu X.\, \mathsf{B} & =_{\mathrm{def}} \forall X.\, (\mathsf{B} \to X) \to X & (X \text{ +ve in } \mathsf{B}) \\
\nu X.\, \mathsf{B} & =_{\mathrm{def}} \exists X.\, (X \to \mathsf{B}) \times X & (X \text{ +ve in } \mathsf{B})
\end{array}
$$

Fig. 2. Definable value types

$$
\begin{array}{rll}
!\mathsf{B} & =_{\mathrm{def}} \forall \underline{X}.\, (\mathsf{B} \to \underline{X}) \to \underline{X} & (\underline{X} \notin \mathrm{FTV}(\mathsf{B})) \\
1^{\circ} & =_{\mathrm{def}} \forall \underline{X}.\, 0 \to \underline{X} & \\
\underline{\mathsf{A}} \times^{\circ} \underline{\mathsf{B}} & =_{\mathrm{def}} \forall \underline{X}.\, ((\underline{\mathsf{A}} \multimap \underline{X}) + (\underline{\mathsf{B}} \multimap \underline{X})) \to \underline{X} & (\underline{X} \notin \mathrm{FTV}(\underline{\mathsf{A}},\underline{\mathsf{B}})) \\
0^{\circ} & =_{\mathrm{def}} \forall \underline{X}.\, \underline{X} & \\
\underline{\mathsf{A}} \oplus \underline{\mathsf{B}} & =_{\mathrm{def}} \forall \underline{X}.\, (\underline{\mathsf{A}} \multimap \underline{X}) \to (\underline{\mathsf{B}} \multimap \underline{X}) \to \underline{X} & (\underline{X} \notin \mathrm{FTV}(\underline{\mathsf{A}},\underline{\mathsf{B}})) \\
\mathsf{B} \cdot \underline{\mathsf{A}} & =_{\mathrm{def}} \forall \underline{X}.\, (\mathsf{B} \to \underline{\mathsf{A}} \multimap \underline{X}) \to \underline{X} & (\underline{X} \notin \mathrm{FTV}(\mathsf{B},\underline{\mathsf{A}})) \\
\exists^{\circ} X.\, \underline{\mathsf{A}} & =_{\mathrm{def}} \forall \underline{Y}.\, (\forall X.\, (\underline{\mathsf{A}} \multimap \underline{Y})) \to \underline{Y} & (\underline{Y} \notin \mathrm{FTV}(\underline{\mathsf{A}})) \\
\exists^{\circ} \underline{X}.\, \underline{\mathsf{A}} & =_{\mathrm{def}} \forall \underline{Y}.\, (\forall \underline{X}.\, (\underline{\mathsf{A}} \multimap \underline{Y})) \to \underline{Y} & (\underline{Y} \notin \mathrm{FTV}(\underline{\mathsf{A}})) \\
\mu^{\circ} \underline{X}.\, \underline{\mathsf{A}} & =_{\mathrm{def}} \forall \underline{X}.\, (\underline{\mathsf{A}} \multimap \underline{X}) \to \underline{X} & (\underline{X} \text{ +ve in } \underline{\mathsf{A}}) \\
\nu^{\circ} \underline{X}.\, \underline{\mathsf{A}} & =_{\mathrm{def}} \exists^{\circ} \underline{X}.\, (\underline{X} \multimap \underline{\mathsf{A}}) \cdot \underline{X} & (\underline{X} \text{ +ve in } \underline{\mathsf{A}})
\end{array}
$$

Fig. 3. Definable computation types

Since value types extend second-order lambda-calculus, the polymorphic type encodings known from that case can be used for type encodings on value types in PE. Figure 2 recalls these type encodings and also shows how to encode existential quantification over computation types. Note that the encodings of inductive and coinductive types require a positive polarity of the type variable X. This notion is defined in the standard way, cf. Section 5.

Figure 3 describes polymorphic encodings of a number of constructions on computation types. The first of these is the free computation type $!\mathsf{B}$ on a value type B. This plays the role of the monad in Moggi's computational lambda-calculus [6], or more precisely of the F constructor in CBPV [3] (for further details, see [4]). The next constructions are unit, product, initial object and binary coproduct of computation types. The type $\mathsf{B} \cdot \underline{\mathsf{A}}$ is the B-fold copower of $\underline{\mathsf{A}}$, and thus can be thought of as a coproduct $\coprod_{x \in \mathsf{B}} \underline{\mathsf{A}}$ of computation types indexed by a value type. The remaining constructions are existential quantification over value types and computation types, packaged up as computation types, and inductive and coinductive computation types. We remark that the somewhat exotic-looking types appearing in this figure do have applications. For example,

in forthcoming work, we shall demonstrate an application to giving a (linear) continuation-passing translation of Levy's CBPV.

In Section 3 below we formulate a logic for reasoning about relational parametricity in PE. The main applications of this logic, in Sections 4 and 5, will be to verify the correctness of a selection of the above type encodings.

3 The logic

This section presents the first main contribution of the paper, our logic for reasoning about parametricity in PE. As mentioned in the introduction, this logic has been extracted as a formalization of the reasoning principles validated by the relationally-parametric models of PE described in [4]. The purpose of this paper, however, is to give a self-contained presentation of the logic without reference to [4]. The idea is that the logic can be understood independently of its (somewhat convoluted) models.

In order to be well-typed, propositions are defined in contexts of relation variables and term variables, denoted Θ and Γ respectively in the meta-notation. As mentioned in the introduction, the logic has two classes of relations: *value relations* between value types and *computation relations* between computation types. We use the notations $\mathsf{Rel}_v(\mathsf{A},\mathsf{B})$ and $\mathsf{Rel}_c(\underline{\mathsf{A}},\underline{\mathsf{B}})$ for the collections of all value relations between A and B and all computation relations between $\underline{\mathsf{A}}$ and $\underline{\mathsf{B}}$ respectively. The formation rules for propositions are given in Figure 4. In the figure, the notation $\mathsf{Rel}_{\_}(\mathsf{A},\mathsf{B})$ is used in some rules. In these cases the rule holds for both value relations and computation relations, and so is a shorthand for two rules. Note that we only include connectives and quantifiers from the negative fragment of intuitionistic logic. Although the others could be included in principle, we shall not need them, and so omit them for space reasons.

The formation rules for relations are given in Figure 5. Relations are closed under: conjunctions, universal quantification and under implications whose antecendent does not depend on the variables being related. This last restriction is motivated by the models considered in [4]. A similar condition is required on *admissible relations* in [2]. For the purposes of the present paper, this condition should just be accepted as a syntactic condition that needs to be adhered to when using the logic.

Lemma 1. *1. If* $\rho\colon \mathsf{Rel}_c(\underline{\mathsf{A}},\underline{\mathsf{B}})$ *in some context then also* $\rho\colon \mathsf{Rel}_v(\underline{\mathsf{A}},\underline{\mathsf{B}})$ *in the same context.*

2. If $\Gamma \mid - \vdash f\colon \mathsf{A} \to \mathsf{B}$ *then* $\Theta\,;\,\Gamma \vdash (x\colon \mathsf{A}, y\colon \mathsf{B}).\, f(x) = y : \mathsf{Rel}_v(\mathsf{A},\mathsf{B})$ *for any relational context* Θ.

3. If $\Gamma \mid - \vdash g\colon \underline{\mathsf{A}} \multimap \underline{\mathsf{B}}$ *then* $\Theta\,;\,\Gamma \vdash (x\colon \underline{\mathsf{A}}, y\colon \underline{\mathsf{B}}).\, g(x) = y : \mathsf{Rel}_c(\underline{\mathsf{A}},\underline{\mathsf{B}})$.

We write $\langle f\rangle$ and $\langle g\rangle$ for the relations of items 2 and 3 in the lemma, and call these relations graphs. We use eq_{A} for the graph of the identity function on A.

Since relations ρ are always of the form $(x\colon \mathsf{A}, y\colon \mathsf{B}).\,\phi$ we can use the meta-notation $\rho(t,u)$ for $\phi[t,u/x,y]$ whenever $\Gamma \mid - \vdash t\colon \mathsf{A}$ and $\Gamma \mid - \vdash u\colon \mathsf{B}$.

$$\frac{\Gamma \mid - \vdash t \colon \mathsf{A} \qquad \Gamma \mid - \vdash u \colon \mathsf{A}}{\Theta \,;\, \Gamma \vdash t =_{\mathsf{A}} u : \mathsf{Prop}} \qquad \frac{\Gamma \mid - \vdash t \colon \mathsf{A} \qquad \Gamma \mid - \vdash u \colon \mathsf{B} \qquad R \colon \mathsf{Rel}_{-}(\mathsf{A},\mathsf{B}) \in \Gamma}{\Theta \,;\, \Gamma \vdash R(t,u) : \mathsf{Prop}}$$

$$\frac{\Theta \,;\, \Gamma \vdash \phi : \mathsf{Prop} \qquad \Theta \,;\, \Gamma \vdash \psi : \mathsf{Prop}}{\Theta \,;\, \Gamma \vdash \phi \square \psi : \mathsf{Prop}}\ \square \in \{\wedge, \supset\} \qquad \frac{\Theta \,;\, \Gamma, x \colon \mathsf{A} \vdash \phi : \mathsf{Prop}}{\Theta \,;\, \Gamma \vdash \forall x \colon \mathsf{A}.\, \phi : \mathsf{Prop}}$$

$$\frac{\Theta, R \colon \mathsf{Rel}_{-}(\mathsf{A},\mathsf{B}) \,;\, \Gamma \vdash \phi : \mathsf{Prop}}{\Theta \,;\, \Gamma \vdash \forall R \colon \mathsf{Rel}_{-}(\mathsf{A},\mathsf{B}).\, \phi : \mathsf{Prop}} \qquad \frac{\Theta \,;\, \Gamma \vdash \phi : \mathsf{Prop}}{\Theta \,;\, \Gamma \vdash \forall \underline{X}.\, \phi : \mathsf{Prop}}\ (\star) \qquad \frac{\Theta \,;\, \Gamma \vdash \phi : \mathsf{Prop}}{\Theta \,;\, \Gamma \vdash \forall X.\, \phi : \mathsf{Prop}}\ (\star)$$

Fig. 4. Typing rules for propositions. Here $(\star)$ is the side condition $X \notin \mathrm{FTV}(\Theta, \Gamma)$ and $-$ ranges over $\{v, c\}$.

$$\frac{\Gamma, x \colon \mathsf{A} \mid - \vdash t \colon \mathsf{C} \qquad \Gamma, y \colon \mathsf{B} \mid - \vdash u \colon \mathsf{C}}{\Theta \,;\, \Gamma \vdash (x \colon \mathsf{A}, y \colon \mathsf{B}).\, t = u \colon \mathsf{Rel}_{v}(\mathsf{A},\mathsf{B})} \qquad \frac{\Gamma, x \colon \mathsf{A}' \mid - \vdash t \colon \mathsf{A} \qquad \Gamma, y \colon \mathsf{B}' \mid - \vdash u \colon \mathsf{B}}{\Theta, R \colon \mathsf{Rel}_{-}(\mathsf{A},\mathsf{B}) \,;\, \Gamma \vdash (x \colon \mathsf{A}', y \colon \mathsf{B}').\, R(t,u) \colon \mathsf{Rel}_{v}(\mathsf{A}',\mathsf{B}')}$$

$$\frac{\Gamma \mid x \colon \underline{\mathsf{A}} \vdash t \colon \underline{\mathsf{C}} \qquad \Gamma \mid y \colon \underline{\mathsf{B}} \vdash u \colon \underline{\mathsf{C}}}{\Theta \,;\, \Gamma \vdash (x \colon \underline{\mathsf{A}}, y \colon \underline{\mathsf{B}}).\, t = u \colon \mathsf{Rel}_{c}(\underline{\mathsf{A}},\underline{\mathsf{B}})} \qquad \frac{\Gamma \mid x \colon \underline{\mathsf{A}}' \vdash t \colon \underline{\mathsf{A}} \qquad \Gamma \mid y \colon \underline{\mathsf{B}}' \vdash u \colon \underline{\mathsf{B}}}{\Theta, \underline{R} \colon \mathsf{Rel}_{c}(\underline{\mathsf{A}},\underline{\mathsf{B}}) ; \Gamma \vdash (x \colon \underline{\mathsf{A}}', y \colon \underline{\mathsf{B}}').\, \underline{R}(t,u) \colon \mathsf{Rel}_{c}(\underline{\mathsf{A}}',\underline{\mathsf{B}}')}$$

$$\frac{\Theta \,;\, \Gamma, z \colon \mathsf{C} \vdash (x \colon \mathsf{A}, y \colon \mathsf{B}).\, \phi \colon \mathsf{Rel}_{-}(\mathsf{A},\mathsf{B})}{\Theta \,;\, \Gamma \vdash (x \colon \mathsf{A}, y \colon \mathsf{B}).\, \forall z \colon \mathsf{C}.\, \phi \colon \mathsf{Rel}_{-}(\mathsf{A},\mathsf{B})} \qquad \frac{\Theta, R \colon \mathsf{Rel}_{=}(\mathsf{C},\mathsf{C}') \,;\, \Gamma \vdash (x \colon \mathsf{A}, y \colon \mathsf{B}).\, \phi \colon \mathsf{Rel}_{-}(\mathsf{A},\mathsf{B})}{\Theta \,;\, \Gamma \vdash (x \colon \mathsf{A}, y \colon \mathsf{B}).\, \forall R \colon \mathsf{Rel}_{=}(\mathsf{C},\mathsf{C}').\, \phi : \mathsf{Rel}_{-}(\mathsf{A},\mathsf{B})}$$

$$\frac{\Theta \,;\, \Gamma \vdash (x \colon \mathsf{A}, y \colon \mathsf{B}).\, \phi : \mathsf{Rel}_{-}(\mathsf{A},\mathsf{B})}{\Theta \,;\, \Gamma \vdash (x \colon \mathsf{A}, y \colon \mathsf{B}).\, \forall X.\, \phi : \mathsf{Rel}_{-}(\mathsf{A},\mathsf{B})} \qquad \frac{\Theta \,;\, \Gamma \vdash (x \colon \mathsf{A}, y \colon \mathsf{B}).\, \phi : \mathsf{Rel}_{-}(\mathsf{A},\mathsf{B})}{\Theta \,;\, \Gamma \vdash (x \colon \mathsf{A}, y \colon \mathsf{B}).\, \forall \underline{X}.\, \phi : \mathsf{Rel}_{-}(\mathsf{A},\mathsf{B})}$$

$$\frac{\Theta \,;\, \Gamma \vdash \psi : \mathsf{Prop} \qquad \Theta \,;\, \Gamma \vdash (x \colon \mathsf{A}, y \colon \mathsf{B}).\, \phi : \mathsf{Rel}_{-}(\mathsf{A},\mathsf{B})}{\Theta \,;\, \Gamma \vdash (x \colon \mathsf{A}, y \colon \mathsf{B}).\, \psi \supset \phi : \mathsf{Rel}_{-}(\mathsf{A},\mathsf{B})}$$

Fig. 5. Typing rules for relations. In these rules $-, =$ range over $\{v, c\}$.

Similarly we can write $\rho^{\mathrm{op}} \colon \mathsf{Rel}_{-}(\mathsf{B},\mathsf{A})$ for $(y \colon \mathsf{B}, x \colon \mathsf{A}).\, \phi$. If ρ is a value relation then so is ρ^{op}, and likewise for computation relations.

Deduction sequents are written on the form $\Theta \,;\, \Gamma \mid \Phi \vdash \psi$ where Φ is a finite set of formulas. A deduction sequent is well-formed if $\Theta \,;\, \Gamma \vdash \psi : \mathsf{Prop}$ and $\Theta \,;\, \Gamma \vdash \phi_i : \mathsf{Prop}$ for all ϕ_i in Φ, and we shall assume well-formedness whenever writing a deduction sequent. The rules for deduction in the logic are presented in Figure 6, to which should be added the rules for β and η equality as in Figure 7, and the usual rules for implication, conjunction, which we have omitted for reasons of space. The rules for equality implement a congruence relation on terms (the congruence rules not explicit in Figure 6 can be derived from the equality elimination rule).

An important application of the logic is to prove equalities between terms. For terms $\Gamma \mid \Delta \vdash s \colon \mathsf{A}$ and $\Gamma \mid \Delta \vdash t \colon \mathsf{A}$, we write

$$\Gamma \mid \Delta \vdash s = t \colon \mathsf{A}$$

$$\frac{\Theta\,;\,\Gamma, x\colon \mathsf{A}\,|\Phi\,\vdash\,\psi}{\Theta\,;\,\Gamma\,|\Phi\,\vdash\,\forall x\colon \mathsf{A}.\,\psi}\; x \notin \mathrm{FV}(\Phi) \qquad \frac{\Theta\,;\,\Gamma\,|\Phi\,\vdash\,\forall x\colon \mathsf{A}.\,\psi \quad \Gamma\,|-\,\vdash\,t\colon \mathsf{A}}{\Theta\,;\,\Gamma\,|\Phi\,\vdash\,\psi[t/x]}$$

$$\frac{\Theta, R\colon \mathsf{Rel}_{-}(\mathsf{A},\mathsf{B})\,;\,\Gamma\,|\Phi\,\vdash\,\psi}{\Theta\,;\,\Gamma\,|\Phi\,\vdash\,\forall R\colon \mathsf{Rel}_{-}(\mathsf{A},\mathsf{B}).\,\psi}\; - \in \{v, c\}$$

$$\frac{\Theta\,;\,\Gamma\,|\Phi\,\vdash\,\forall R\colon \mathsf{Rel}_{-}(\mathsf{A},\mathsf{B}).\,\psi \quad \Theta\,;\,\Gamma\,\vdash\,(x\colon \mathsf{A}, y\colon \mathsf{B}).\,\phi : \mathsf{Rel}_{-}(\mathsf{A},\mathsf{B})}{\Theta\,;\,\Gamma\,|\Phi\,\vdash\,\psi[\phi[t,u/x,y]/R(t,u)]}\; - \in \{v, c\}$$

$$\frac{\Theta\,;\,\Gamma\,|\Phi\,\vdash\,\psi}{\Theta\,;\,\Gamma\,|\Phi\,\vdash\,\forall X.\,\psi}\; X \notin \mathrm{FTV}(\Theta,\Gamma,\Phi) \qquad \frac{\Theta\,;\,\Gamma\,|\Phi\,\vdash\,\forall X.\,\psi}{\Theta\,;\,\Gamma\,|\Phi\,\vdash\,\psi[\mathsf{A}/X]}$$

$$\frac{\Theta\,;\,\Gamma\,|\Phi\,\vdash\,\psi}{\Theta\,;\,\Gamma\,|\Phi\,\vdash\,\forall \underline{X}.\,\psi}\; \underline{X} \notin \mathrm{FTV}(\Theta,\Gamma,\Phi) \qquad \frac{\Theta\,;\,\Gamma\,|\Phi\,\vdash\,\forall \underline{X}.\,\psi}{\Theta\,;\,\Gamma\,|\Phi\,\vdash\,\psi[\underline{\mathsf{A}}/\underline{X}]}$$

$$\frac{\Gamma\,|-\,\vdash\,t\colon \mathsf{A}}{\Theta\,;\,\Gamma\,|\Phi\,\vdash\,t = t} \qquad \frac{\Theta\,;\,\Gamma\,|\Phi\,\vdash\,t = u \quad \Theta\,;\,\Gamma\,|\Phi\,\vdash\,\phi[t/x]}{\Theta\,;\,\Gamma\,|\Phi\,\vdash\,\phi[u/x]}$$

$$\frac{\Gamma,\, x\colon \mathsf{B}\,|-\,\vdash\,t, u\colon \mathsf{C} \quad \Theta\,;\,\Gamma,\, x\colon \mathsf{B}\,|\Phi\,\vdash\,t = u}{\Theta\,;\,\Gamma\,|\Phi\,\vdash\,\lambda x\colon \mathsf{B}.\,t = \lambda x\colon \mathsf{B}.\,u}\; x \notin \mathrm{FV}(\Phi)$$

$$\frac{\Gamma\,|-\,\vdash\,t, u\colon \mathsf{B} \quad \Theta\,;\,\Gamma\,|\Phi\,\vdash\,t = u}{\Theta\,;\,\Gamma\,|\Phi\,\vdash\,\Lambda X.\,t = \Lambda X.\,u}\; X \notin \mathrm{FTV}(\Gamma,\Delta,\Phi)$$

$$\frac{\Gamma\,|x\colon \underline{\mathsf{A}}\,\vdash\,t, u\colon \underline{\mathsf{B}} \quad \Theta\,;\,\Gamma,\, x\colon \underline{\mathsf{A}}\,|\Phi\,\vdash\,t = u}{\Theta\,;\,\Gamma\,|\Phi\,\vdash\,\lambda^{\circ} x\colon \underline{\mathsf{A}}.\,t = \lambda^{\circ} x\colon \underline{\mathsf{A}}.\,u}\; x \notin \mathrm{FV}(\Phi)$$

$$\frac{\Gamma\,|-\,\vdash\,t, u\colon \mathsf{B} \quad \Theta\,;\,\Gamma\,|\Phi\,\vdash\,t = u}{\Theta\,;\,\Gamma\,|\Phi\,\vdash\,\Lambda \underline{X}.\,t = \Lambda \underline{X}.\,u}\; \underline{X} \notin \mathrm{FTV}(\Gamma,\Delta,\Phi)$$

Fig. 6. Deduction rules.

(although we shall often omit the type A) as notation for the deduction sequent $-\,;\,\Gamma,\,\Delta\,\vdash\,t = u$. Thus $\Gamma\,|\,\Delta\,\vdash\,s = t\colon \mathsf{A}$ and $\Gamma, \Delta\,|-\,\vdash\,s = t\colon \mathsf{A}$ are equivalent. This corresponds to the faithfulness of the forgetful functor from computation types to value types in the semantic models of [4]. A related fact, is that the canonical map of type $(\underline{\mathsf{A}} \multimap \underline{\mathsf{B}}) \to \underline{\mathsf{A}} \to \underline{\mathsf{B}}$ given by the term $\lambda f\colon \underline{\mathsf{A}} \multimap \underline{\mathsf{B}}.\,\lambda x\colon \underline{\mathsf{A}}.\,f(x)$ is injective, which is derivable using the lemma below.

Lemma 2. *The following extensionality schemas are provable in the logic.*

$$\forall f, g\colon \mathsf{A} \to \mathsf{B}.\,(\forall x\colon \mathsf{A}.\,f(x) = g(x)) \supset f = g$$
$$\forall f, g\colon \underline{\mathsf{A}} \multimap \underline{\mathsf{B}}.\,(\forall x\colon \underline{\mathsf{A}}.\,f(x) = g(x)) \supset f = g$$
$$\forall x, y\colon (\forall X.\,\mathsf{A}).\,(\forall X.\,x\,X = y\,X) \supset x = y$$
$$\forall x, y\colon (\forall \underline{X}.\,\mathsf{A}).\,(\forall \underline{X}.\,x\,\underline{X} = y\,\underline{X}) \supset x = y$$

$$
\begin{array}{rrl}
(\lambda x\colon \mathsf{A}.\, t)(u) = t[u/x] & \lambda x\colon \mathsf{A}.\, t(x) = t & \text{if } t\colon \mathsf{A} \to \mathsf{B} \text{ and } x \notin \mathrm{FV}(t) \\
(\lambda^\circ x\colon \underline{\mathsf{A}}.\, t)(u) = t[u/x] & \lambda^\circ x\colon \underline{\mathsf{A}}.\, t(x) = t & \text{if } t\colon \underline{\mathsf{A}} \multimap \underline{\mathsf{B}} \text{ and } x \notin \mathrm{FV}(t) \\
(\Lambda \underline{X}.\, t)\, \underline{\mathsf{A}} = t[\underline{\mathsf{A}}/\underline{X}] & \Lambda \underline{Y}.\, t\, \underline{Y} = t & \text{if } t\colon \forall \underline{X}.\, \mathsf{A} \text{ and } \underline{Y} \notin \mathrm{FTV}(t) \\
(\Lambda X.\, t)\, \mathsf{A} = t[\mathsf{A}/X] & \Lambda Y.\, t\, Y = t & \text{if } t\colon \forall X.\, \mathsf{A} \text{ and } Y \notin \mathrm{FTV}(t)
\end{array}
$$

Fig. 7. β, η rules for PE.

$$
\begin{aligned}
X_i[\boldsymbol{\rho}, \underline{\boldsymbol{\rho}}] &= \rho_i \\
\underline{X}_j[\boldsymbol{\rho}, \underline{\boldsymbol{\rho}}] &= \underline{\rho}_j \\
(\mathsf{A} \to \mathsf{B})[\boldsymbol{\rho}, \underline{\boldsymbol{\rho}}] &= (f\colon (\mathsf{A} \to \mathsf{B})(\mathbf{C}, \underline{\mathbf{C}}), g\colon (\mathsf{A} \to \mathsf{B})(\mathbf{C}', \underline{\mathbf{C}}'))\,.\, \forall x\colon \mathsf{A}(\mathbf{C}, \underline{\mathbf{C}}), \\
&\qquad \forall y\colon \mathsf{A}(\mathbf{C}', \underline{\mathbf{C}}')\,.\, \mathsf{A}[\boldsymbol{\rho}, \underline{\boldsymbol{\rho}}](x, y) \supset \mathsf{B}[\boldsymbol{\rho}, \underline{\boldsymbol{\rho}}](f(x), g(y)) \\
(\underline{\mathsf{A}} \multimap \underline{\mathsf{B}})[\boldsymbol{\rho}, \underline{\boldsymbol{\rho}}] &= (f\colon (\underline{\mathsf{A}} \multimap \underline{\mathsf{B}})(\mathbf{C}, \underline{\mathbf{C}}), g\colon (\underline{\mathsf{A}} \multimap \underline{\mathsf{B}})(\mathbf{C}', \underline{\mathbf{C}}'))\,.\, \forall x\colon \underline{\mathsf{A}}(\mathbf{C}, \underline{\mathbf{C}}), \\
&\qquad \forall y\colon \underline{\mathsf{B}}(\mathbf{C}', \underline{\mathbf{C}}')\,.\, \underline{\mathsf{A}}[\boldsymbol{\rho}, \underline{\boldsymbol{\rho}}](x, y) \supset \underline{\mathsf{B}}[\boldsymbol{\rho}, \underline{\boldsymbol{\rho}}](f(x), g(y)) \\
(\forall X.\, \mathsf{A})[\boldsymbol{\rho}, \underline{\boldsymbol{\rho}}] &= (x\colon \forall X.\, \mathsf{A}[\mathbf{C}, \underline{\mathbf{C}}], y\colon \forall X.\, \mathsf{A}[\mathbf{C}, \underline{\mathbf{C}}])\,.\, \forall Y, Z\,.\, \forall R\colon \mathsf{Rel}_v(Y, Z)\,.\, \mathsf{A}[\boldsymbol{\rho}, \underline{\boldsymbol{\rho}}, R](x\, Y, y\, Z) \\
(\forall \underline{X}.\, \mathsf{A})[\boldsymbol{\rho}, \underline{\boldsymbol{\rho}}] &= (x\colon \forall \underline{X}.\, \mathsf{A}[\mathbf{C}, \underline{\mathbf{C}}], y\colon \forall \underline{X}.\, \mathsf{A}[\mathbf{C}, \underline{\mathbf{C}}])\,.\, \forall \underline{Y}, \underline{Z}\,.\, \forall \underline{R}\colon \mathsf{Rel}_c(\underline{Y}, \underline{Z})\,.\, \mathsf{A}[\boldsymbol{\rho}, \underline{\boldsymbol{\rho}}, \underline{R}](x\, \underline{Y}, y\, \underline{Z})
\end{aligned}
$$

Fig. 8. Relational interpretation of types.

We now come to the crucial *relational interpretation* of types, needed to define relational parametricity. Suppose A is a type such that $\mathrm{FTV}(\mathsf{A}) \subseteq \{\boldsymbol{X}, \underline{\boldsymbol{X}}\}$ (using bold font for vectors), and $\boldsymbol{\rho}$ and $\underline{\boldsymbol{\rho}}$ are vectors of relations of the same lengths as $\boldsymbol{X}$ and $\underline{\boldsymbol{X}}$ respectively such that $\Theta\,;\, \Gamma \vdash \rho_i : \mathsf{Rel}_v(\mathsf{C}_i, \mathsf{C}'_i)$ for each i indexing an element of $\boldsymbol{X}$, and $\Theta\,;\, \Gamma \vdash \underline{\rho}_j : \mathsf{Rel}_c(\underline{\mathsf{C}}_j, \underline{\mathsf{C}}'_j)$ for each j indexing an element of $\underline{\boldsymbol{X}}$. We define $\mathsf{A}[\boldsymbol{\rho}, \underline{\boldsymbol{\rho}}/\boldsymbol{X}, \underline{\boldsymbol{X}}] \colon \mathsf{Rel}_v(\mathsf{A}[\mathbf{C}, \underline{\mathbf{C}}/\boldsymbol{X}, \underline{\boldsymbol{X}}], \mathsf{A}[\mathbf{C}', \underline{\mathbf{C}}'/\boldsymbol{X}, \underline{\boldsymbol{X}}])$ by structural induction on A as in Figure 8, using the short notation $\mathsf{A}[\boldsymbol{\rho}, \underline{\boldsymbol{\rho}}]$ for $\mathsf{A}[\boldsymbol{\rho}, \underline{\boldsymbol{\rho}}/\boldsymbol{X}, \underline{\boldsymbol{X}}]$ and $\mathsf{A}(\mathbf{C}, \underline{\mathbf{C}})$ for $\mathsf{A}[\mathbf{C}, \underline{\mathbf{C}}/\boldsymbol{X}, \underline{\boldsymbol{X}}]$.

Lemma 3. *If $\underline{\mathsf{A}}$ is a computation type then $\underline{\mathsf{A}}[\boldsymbol{\rho}, \underline{\boldsymbol{\rho}}]$ is a computation relation.*

As our axiom for parametricity we shall take a version of Reynolds' identity extension schema [10] adapted to our setting. Using the shorthand notation $\rho \equiv \rho'$ for $\forall x, y.\, \rho(x, y) \leftrightarrows \rho'(x, y)$ this can be stated as:

$$
\mathsf{A}[eq_{\mathbf{B}}, eq_{\underline{\mathbf{B}}}] \equiv eq_{\mathsf{A}[\mathbf{B}, \underline{\mathbf{B}}]} \tag{1}
$$

where A ranges over all value types such that $\mathrm{FTV}(\mathsf{A}) \subseteq \{\boldsymbol{X}, \underline{\boldsymbol{X}}\}$ and $\mathbf{B}$ and $\underline{\mathbf{B}}$ range over all vectors of value types and computation types respectively (open as well as closed).

Lemma 4. *Identity extension (1) is equivalent to the two parametricity schemas:*

$$\forall x\colon (\forall Y.\, \mathsf{A}(\mathbf{B}, \underline{\mathbf{B}}, Y)).\, \forall Y, Z, R\colon \mathsf{Rel}_v(Y, Z).\, \mathsf{A}[eq_{\mathbf{B}}, eq_{\underline{\mathbf{B}}}, R](x(Y), x(Z))$$
$$\forall x\colon (\forall \underline{Y}.\, \mathsf{A}'(\mathbf{B}, \underline{\mathbf{B}}, \underline{Y})).\, \forall \underline{Y}, \underline{Z}, \underline{R}\colon \mathsf{Rel}_c(\underline{Y}, \underline{Z}).\, \mathsf{A}'[eq_{\mathbf{B}}, eq_{\underline{\mathbf{B}}}, \underline{R}](x(\underline{Y}), x(\underline{Z}))$$

where A, A' *range over types with* $FTV(\mathsf{A}) \subseteq \{\boldsymbol{X}, \underline{\boldsymbol{X}}, Y\}$ *and* $FTV(\mathsf{A}') \subseteq \{\boldsymbol{X}, \underline{\boldsymbol{X}}, \underline{Y}\}$

In the case of parametricity in the second-order lambda-calculus, the equivalence asserted by Lemma 4 is well known. The proof in our setting is similar. In one direction, the parametricity schemas are special cases of the identity extension schema in the case of polymorphic types. The other direction is proved by induction over the structure of types.

Lemma 5 (Logical relations). *Suppose* $\Gamma \mid \Delta \vdash t\colon \mathsf{A}$ *is a valid typing judgement with* $FTV(\Gamma, \Delta, t, \mathsf{A}) \subseteq \{\boldsymbol{X}, \underline{\boldsymbol{X}}\}$ *and suppose we are given vectors of relations* $\boldsymbol{\rho}\colon \mathsf{Rel}_v(\mathbf{C}, \mathbf{C}')$ *and* $\underline{\boldsymbol{\rho}}\colon \mathsf{Rel}_c(\underline{\mathbf{C}}, \underline{\mathbf{C}}')$*. Suppose we are further given* $s_i\colon \mathsf{B}_i(\mathbf{C}, \underline{\mathbf{C}})$ *and* $s_i'\colon \mathsf{B}_i(\mathbf{C}', \underline{\mathbf{C}}')$ *for each* $x_i\colon \mathsf{B}_i$ *in* Γ*, and if* $\Delta = x_{n+1}\colon \underline{\mathsf{B}}_{n+1}$ *is non-empty also* $s_{n+1}\colon \mathsf{B}_{n+1}(\mathbf{C}, \underline{\mathbf{C}})$ *and* $s_{n+1}'\colon \mathsf{B}_{n+1}(\mathbf{C}', \underline{\mathbf{C}}')$*. If* $\mathsf{B}_i[\boldsymbol{\rho}, \underline{\boldsymbol{\rho}}](s_i, s_i')$ *for all* i*, then*

$$\mathsf{A}[\boldsymbol{\rho}, \underline{\boldsymbol{\rho}}](t[\boldsymbol{s}, \mathbf{C}, \underline{\mathbf{C}}/\boldsymbol{x}, \boldsymbol{X}, \underline{\boldsymbol{X}}], t[\boldsymbol{s}', \mathbf{C}', \underline{\mathbf{C}}'/\boldsymbol{x}, \boldsymbol{X}, \underline{\boldsymbol{X}}])$$

In the sequel, we apply the logic defined in this section to verify properties of PE. In doing so, we call the underlying logic, without the identity extension schema, L; and we write L+P for the logic with the identity extension schema (equivalently parametricity schemas) added.

4 Verifying polymorphic type encodings

In this section, we apply the logic to verify the correctness of a selection of the datatype encodings presented in Section 2. Our style will be arguments in an informal style, including as much detail as space permits, but to ensure that the arguments are always directly formalizable.

The value type encodings of Figure 2, can be verified essentially as in Plotkin and Abadi's logic [7] (see also [1]). Nevertheless, we briefly discuss the the case of coproducts, as it serves to illustrate a subtlety introduced by the stoup.

The type $\mathsf{A}+\mathsf{B}$ supports derived introduction and elimination rules as follows.

$$\frac{\Gamma \mid - \vdash t\colon \mathsf{A}}{\Gamma \mid - \vdash \mathrm{in}_1(t)\colon \mathsf{A}+\mathsf{B}} \qquad \frac{\Gamma \mid - \vdash u\colon \mathsf{B}}{\Gamma \mid - \vdash \mathrm{in}_2(u)\colon \mathsf{A}+\mathsf{B}}$$
$$\frac{\Gamma, x\colon \mathsf{A} \mid \Delta \vdash u\colon \mathsf{C} \quad \Gamma, y\colon \mathsf{B} \mid \Delta \vdash u'\colon \mathsf{C} \quad \Gamma \mid - \vdash t\colon \mathsf{A}+\mathsf{B}}{\Gamma \mid \Delta \vdash \mathrm{case}\, t\, \mathrm{of}\, \mathrm{in}_1(x).\, u;\ \mathrm{in}_2(y).\, u'\colon \mathsf{C}} \tag{2}$$

Here, the left and right inclusions are defined as expected:

$$\mathrm{in}_1(t) =_{\mathrm{def}} \Lambda X.\, \lambda f\colon \mathsf{A} \to X.\, \lambda g\colon \mathsf{B} \to X.\, f(t)$$
$$\mathrm{in}_2(u) =_{\mathrm{def}} \Lambda X.\, \lambda f\colon \mathsf{A} \to X.\, \lambda g\colon \mathsf{B} \to X.\, g(u)$$

But the definition of the case construction depends on the stoup. If the stoup is empty then

$$\mathrm{case}\, t \,\mathrm{of}\, \mathrm{in}_1(x).\, u;\ \mathrm{in}_2(y).\, u' =_{\mathrm{def}} t\, \mathsf{C}\, (\lambda x\colon \mathsf{A}.\, u)\, (\lambda y\colon \mathsf{A}.\, u')$$

but if it is non-empty, say $\Delta = z\colon \underline{\mathsf{C}}'$ then $\mathrm{case}\, t \,\mathrm{of}\, \mathrm{in}_1(x).\, u;\ \mathrm{in}_2(y).\, u'$ is:

$$(t\, (\mathsf{C}' \multimap \mathsf{C})\, (\lambda x\colon \mathsf{A}.\, \lambda^\circ z\colon \mathsf{C}'.\, u)\, (\lambda y\colon \mathsf{A}.\, \lambda^\circ z\colon \mathsf{C}'.\, u'))\, z\,.$$

That this encoding of coproducts enjoys the expected universal property is captured by the equalities in the theorem below.

Theorem 1. *Suppose u, u' are as in the hypothesis of the elimination rule (2) then* L *proves*

- *If $\Gamma \mid - \vdash t\colon \mathsf{A}$ then $\Gamma \mid \Delta \vdash \mathrm{case}\, \mathrm{in}_1(t) \,\mathrm{of}\, \mathrm{in}_1(x).\, u;\ \mathrm{in}_2(y).\, u' = u[t/x]$*
- *If $\Gamma \mid - \vdash s\colon \mathsf{B}$ then $\Gamma \mid \Delta \vdash \mathrm{case}\, \mathrm{in}_2(s) \,\mathrm{of}\, \mathrm{in}_1(x).\, u;\ \mathrm{in}_2(y).\, u' = u'[s/y]$*

If $\Gamma \mid - \vdash t\colon \mathsf{A} + \mathsf{B}$ and $\Gamma, z\colon \mathsf{A} + \mathsf{B} \mid \Delta \vdash u\colon \mathsf{C}$ then L+P *proves*

$$\Gamma \mid \Delta \vdash \mathrm{case}\, t \,\mathrm{of}\, \mathrm{in}_1(x).\, u[\mathrm{in}_1(x)/z];\ \mathrm{in}_2(y).\, u[\mathrm{in}_2(y)/z] = u[t/z]$$

We omit the proof since it follows the usual argument using relational parametricity, cf. [7, 1]. Instead, we turn to the constructs on computation types, of Figure 3, whose verification makes essential use of computation relations.

Although the type $!\mathsf{A}$, corresponding to Moggi's $T\mathsf{A}$ [6] and Levy's $F\mathsf{A}$ [3], is a particularly important one for effects; we omit the verication of its universal property here, since the argument is given in detail in [4]. There, an informal argument is presented, which is justified in semantic terms. However, every detail of this argument is directly translatable into our logic.

Instead, as our first example, we consider the type $\mathsf{A} \cdot \underline{\mathsf{B}}$, which represents a A-fold *copower* of $\underline{\mathsf{B}}$. Type theoretically, the universal property requires a natural bijection between terms of type $\mathsf{A} \to (\underline{\mathsf{B}} \multimap \underline{\mathsf{C}})$ and terms of type $(\mathsf{A} \cdot \underline{\mathsf{B}}) \multimap \underline{\mathsf{C}}$. The derived introduction and elimination rules for copowers are

$$\frac{\Gamma \mid - \vdash t\colon \mathsf{A} \qquad \Gamma \mid \Delta \vdash s\colon \underline{\mathsf{B}}}{\Gamma \mid \Delta \vdash t \cdot s\colon \mathsf{A} \cdot \underline{\mathsf{B}}} \tag{3}$$

$$\frac{\Gamma, x\colon \mathsf{A} \mid y\colon \underline{\mathsf{B}} \vdash u\colon \underline{\mathsf{C}} \qquad \Gamma \mid \Delta \vdash t\colon \mathsf{A} \cdot \underline{\mathsf{B}}}{\Gamma \mid \Delta \vdash \mathrm{let}\, x \cdot y \,\mathrm{be}\, t \,\mathrm{in}\, u\colon \underline{\mathsf{C}}} \tag{4}$$

Indeed, we define

$$\begin{aligned} t \cdot s &= \Lambda \underline{X}.\, \lambda f\colon \mathsf{A} \to \underline{\mathsf{B}} \multimap \underline{X}.\, f(t)(s) \\ \mathrm{let}\, x \cdot y \,\mathrm{be}\, t \,\mathrm{in}\, u &= t\, \underline{\mathsf{C}}\, (\lambda x\colon \mathsf{A}.\, \lambda^\circ y\colon \underline{\mathsf{B}}.\, u). \end{aligned}$$

Lemma 6. *Suppose t, u are as in the hypothesis of the elimination rule (4) and $\Gamma \mid - \vdash f\colon \underline{\mathsf{C}} \multimap \underline{\mathsf{C}}'$. Then* L+P *proves*

$$\Gamma \mid \Delta \vdash \mathrm{let}\, x \cdot y \,\mathrm{be}\, t \,\mathrm{in}\, f(u) = f(\mathrm{let}\, x \cdot y \,\mathrm{be}\, t \,\mathrm{in}\, u)$$

Proof. Parametricity for t states

$$-\,;\,\Gamma,\Delta \vdash \forall \underline{X},\underline{Y},\underline{R}\colon \mathsf{Rel}_c(\underline{X},\underline{Y}).\,((eq_{\mathsf{A}} \to eq_{\underline{\mathsf{B}}} \multimap \underline{R}) \to \underline{R})(t\,\underline{X}, t\,\underline{Y}) \qquad (5)$$

By definition, $-\,;\,\Gamma \vdash (eq_{\mathsf{A}} \to eq_{\underline{\mathsf{B}}} \multimap \langle f\rangle)(\lambda x\colon \mathsf{A}.\,\lambda^\circ y\colon \underline{\mathsf{B}}.\,u, \lambda x\colon \mathsf{A}.\,\lambda^\circ y\colon \underline{\mathsf{B}}.\,f(u))$. Also, by Lemma 1, $\langle f\rangle$ is a computation relation, Thus, applying (5) we get:

$$-\,;\,\Gamma,\Delta \vdash \langle f\rangle(t\,\underline{\mathsf{C}}\,(\lambda x\colon \mathsf{A}.\,\lambda^\circ y\colon \underline{\mathsf{B}}.\,u), t\,\underline{\mathsf{C}}'\,(\lambda x\colon \mathsf{A}.\,\lambda^\circ y\colon \underline{\mathsf{B}}.\,f(u)))$$

i.e., by definition of the copower let expressions,

$$-\,;\,\Gamma,\Delta \vdash \langle f\rangle(\mathrm{let}\,x\cdot y\,\mathrm{be}\,t\,\mathrm{in}\,u, \mathrm{let}\,x\cdot y\,\mathrm{be}\,t\,\mathrm{in}\,f(u))$$

So, by definition of $\langle f\rangle$,

$$-\,;\,\Gamma,\Delta \vdash \mathrm{let}\,x\cdot y\,\mathrm{be}\,t\,\mathrm{in}\,f(u) = f(\mathrm{let}\,x\cdot y\,\mathrm{be}\,t\,\mathrm{in}\,u)$$

which means that

$$\Gamma \mid \Delta \vdash \mathrm{let}\,x\cdot y\,\mathrm{be}\,t\,\mathrm{in}\,f(u) = f(\mathrm{let}\,x\cdot y\,\mathrm{be}\,t\,\mathrm{in}\,u)$$

is provable as desired. □

Lemma 7. *Suppose* $\Gamma \mid \Delta \vdash t\colon \mathsf{A}\cdot\underline{\mathsf{B}}$ *and* x, y *are fresh. Then* L+P *proves*

$$\Gamma \mid \Delta \vdash \mathrm{let}\,x\cdot y\,\mathrm{be}\,t\,\mathrm{in}\,x\cdot y = t$$

Proof. By extensionality it suffices to prove that if $\underline{X}$ and f are fresh then

$$\Gamma, f\colon \mathsf{A}\to\underline{\mathsf{B}}\multimap\underline{X} \mid \Delta \vdash (\mathrm{let}\,x\cdot y\,\mathrm{be}\,t\,\mathrm{in}\,x\cdot y)\,\underline{X}\,f = t\,\underline{X}\,f.$$

Since

$$\Gamma, f\colon \mathsf{A}\to\underline{\mathsf{B}}\multimap\underline{X} \mid - \vdash \lambda^\circ x\colon \mathsf{A}\cdot\underline{\mathsf{B}}.\,x\,\underline{X}\,f\colon \mathsf{A}\cdot\underline{\mathsf{B}}\multimap\underline{X},$$

by Lemma 6

$$\Gamma, f\colon \mathsf{A}\to\underline{\mathsf{B}}\multimap\underline{X} \mid \Delta \vdash (\mathrm{let}\,x\cdot y\,\mathrm{be}\,t\,\mathrm{in}\,x\cdot y)\,\underline{X}\,f = \mathrm{let}\,x\cdot y\,\mathrm{be}\,t\,\mathrm{in}\,(x\cdot y\,\underline{X}\,f)$$

But

$$\mathrm{let}\,x\cdot y\,\mathrm{be}\,t\,\mathrm{in}\,(x\cdot y\,\underline{X}\,f) = t\,\underline{X}\,(\lambda x\colon \mathsf{A}.\,\lambda^\circ y\colon \underline{\mathsf{B}}.\,x\cdot y\,\underline{X}\,f) = t\,\underline{X}\,f.$$

□

Theorem 2. *If* $\Gamma \mid - \vdash t\colon \mathsf{A}$, $\Gamma \mid \Delta \vdash s\colon \underline{\mathsf{B}}$ *and* $\Gamma, x\colon \mathsf{A} \mid y\colon \underline{\mathsf{B}} \vdash u\colon \underline{\mathsf{C}}$ *then* L *proves*

$$\Gamma \mid \Delta \vdash \mathrm{let}\,x\cdot y\,\mathrm{be}\,t\cdot s\,\mathrm{in}\,u = u[t, s/x, y]$$

If $\Gamma \mid z\colon \mathsf{A}\cdot\underline{\mathsf{B}} \vdash u\colon \underline{\mathsf{C}}$ *and* $\Gamma \mid \Delta \vdash t\colon \mathsf{A}\cdot\underline{\mathsf{B}}$ *then* L+P *proves*

$$\Gamma \mid \Delta \vdash \mathrm{let}\,x\cdot y\,\mathrm{be}\,t\,\mathrm{in}\,u[x\cdot y/z] = u[t/z]$$

Proof. The first part follows from β and η equalities, and the second part from Lemma 6 and Lemma 7. □

This theorem formulates the desired universal property for copower types.

We consider one other example from Figure 3, existential computation types of the form $\exists^\circ \underline{X}.\, \underline{\mathsf{A}}$. The derived introduction and elimination rules are:

$$\frac{\Gamma \mid \Delta \vdash t \colon \underline{\mathsf{A}}[\underline{\mathsf{B}}/\underline{X}]}{\Gamma \mid \Delta \vdash \langle \underline{\mathsf{B}}, t \rangle \colon \exists^\circ \underline{X}.\, \underline{\mathsf{A}}} \tag{6}$$

$$\frac{\Gamma \mid x \colon \underline{\mathsf{A}} \vdash u \colon \underline{\mathsf{B}} \qquad \Gamma \mid \Delta \vdash t \colon \exists^\circ \underline{X}.\, \underline{\mathsf{A}}}{\Gamma \mid \Delta \vdash \mathrm{let}\, \langle \underline{X}, x \rangle \,\mathrm{be}\, t \,\mathrm{in}\, u \colon \underline{\mathsf{B}}}\, \underline{X} \notin \mathrm{FTV}(\underline{\mathsf{B}}) \tag{7}$$

with the relevant term constructors defined by:

$$\langle \underline{\mathsf{B}}, t \rangle = \Lambda \underline{Y}.\, \lambda f \colon (\forall \underline{X}.\, \underline{\mathsf{A}} \multimap \underline{Y}).\, f\, \underline{\mathsf{B}}\, (t)$$
$$\mathrm{let}\, \langle \underline{X}, x \rangle \,\mathrm{be}\, t \,\mathrm{in}\, u = t\, \underline{\mathsf{B}}\, (\Lambda \underline{X}.\, \lambda^\circ x \colon \underline{\mathsf{A}}.\, u)$$

Since correctness argument follows very closely that for copower types, we merely state the relevant lemmas and theorem, omitting the proofs.

Lemma 8. *Suppose t, u are as in the hypothesis of the elimination rule (7) and that $\Gamma \mid - \vdash f \colon \underline{\mathsf{B}} \multimap \underline{\mathsf{B}}'$. Then* L+P *proves*

$$\Gamma \mid \Delta \vdash \mathrm{let}\, \langle \underline{X}, x \rangle \,\mathrm{be}\, t \,\mathrm{in}\, f(u) = f(\mathrm{let}\, \langle \underline{X}, x \rangle \,\mathrm{be}\, t \,\mathrm{in}\, u)$$

Lemma 9. *Suppose $\Gamma \mid \Delta \vdash t \colon \exists^\circ \underline{X}.\, \underline{\mathsf{A}}$ then* L+P *proves*

$$\Gamma \mid \Delta \vdash \mathrm{let}\, \langle \underline{X}, x \rangle \,\mathrm{be}\, t \,\mathrm{in}\, \langle \underline{X}, x \rangle = t$$

Theorem 3. *Suppose $\Gamma \mid \Delta \vdash t \colon \underline{\mathsf{A}}[\underline{\mathsf{B}}/\underline{X}]$ and $\Gamma \mid x \colon \underline{\mathsf{A}} \vdash u \colon \underline{\mathsf{C}}$ then*

- L *proves*
$$\Gamma \mid \Delta \vdash \mathrm{let}\, \langle \underline{X}, x \rangle \,\mathrm{be}\, \langle \underline{\mathsf{B}}, t \rangle \,\mathrm{in}\, u = u[\underline{\mathsf{B}}, t/\underline{X}, x]$$
- *If $\Gamma \mid z \colon \exists^\circ \underline{X}.\, \underline{\mathsf{A}} \vdash s \colon \underline{\mathsf{C}}$ then* L+P *proves*
$$\Gamma \mid \Delta \vdash \mathrm{let}\, \langle \underline{X}, x \rangle \,\mathrm{be}\, t \,\mathrm{in}\, s[\langle \underline{X}, x \rangle / z] = s[t/z]$$

5 Inductive and coinductive types

This final section describes encodings of general inductive and coinductive computation types and verifies the correctness of the latter. To describe the universal properties of these types we need to consider the functorial actions of the type constructors of PE. This is essentially a standard analysis of type structure, adapted to the setting of the two collections of types in PE.

We define positive and negative occurrences of type variables in types in the standard way ($\to$ and $\multimap$ reverse polarity of the type variables occurring on the

$$
\begin{aligned}
Y_i(\boldsymbol{f},\boldsymbol{g},\boldsymbol{h},\boldsymbol{k}) &= g_i \\
\underline{Y}_j(\boldsymbol{f},\boldsymbol{g},\boldsymbol{h},\boldsymbol{k}) &= k_j \\
(\mathsf{A}\to\mathsf{B})(\boldsymbol{f},\boldsymbol{g},\boldsymbol{h},\boldsymbol{k}) &= \lambda l\colon(\mathsf{A}\to\mathsf{B}).\,\mathsf{B}(\boldsymbol{f},\boldsymbol{g},\boldsymbol{h},\boldsymbol{k})\circ l\circ\mathsf{A}(\boldsymbol{g},\boldsymbol{f},\boldsymbol{k},\boldsymbol{h}) \\
(\underline{\mathsf{A}}\multimap\underline{\mathsf{B}})(\boldsymbol{f},\boldsymbol{g},\boldsymbol{h},\boldsymbol{k}) &= \lambda l\colon(\underline{\mathsf{A}}\multimap\underline{\mathsf{B}}).\,\underline{\mathsf{B}}(\boldsymbol{f},\boldsymbol{g},\boldsymbol{h},\boldsymbol{k})\circ l\circ\underline{\mathsf{A}}(\boldsymbol{g},\boldsymbol{f},\boldsymbol{k},\boldsymbol{h}) \\
(\forall Y.\,\mathsf{A})(\boldsymbol{f},\boldsymbol{g},\boldsymbol{h},\boldsymbol{k}) &= \lambda x\colon\forall Y.\,\mathsf{A}.\,\Lambda Y.\,\mathsf{A}(\boldsymbol{f},\boldsymbol{g},\boldsymbol{h},\boldsymbol{k},id_Y,id_Y)(x\,Y) \\
(\forall \underline{Y}.\,\mathsf{A})(\boldsymbol{f},\boldsymbol{g},\boldsymbol{h},\boldsymbol{k}) &= \lambda x\colon\forall \underline{Y}.\,\mathsf{A}.\,\Lambda \underline{Y}.\,\mathsf{A}(\boldsymbol{f},\boldsymbol{g},\boldsymbol{h},\boldsymbol{k},id_{\underline{Y}},id_{\underline{Y}})(x\,\underline{Y})
\end{aligned}
$$

Fig. 9. The functorial interpretation of types

left, and all other type constructors preserve polarity). If A is a value type in which the variables $\boldsymbol{X},\underline{\boldsymbol{X}}$ occur only negatively and the type variables $\boldsymbol{Y},\underline{\boldsymbol{Y}}$ occur only positively, then we can define a term

$$
\begin{aligned}
M_{\mathsf{A}}\colon \forall \boldsymbol{X},\boldsymbol{X'},\boldsymbol{Y},\boldsymbol{Y'},\underline{\boldsymbol{X}},\underline{\boldsymbol{X'}},\underline{\boldsymbol{Y}},\underline{\boldsymbol{Y'}}.\,(\boldsymbol{X'}\to\boldsymbol{X})\to(\boldsymbol{Y}\to\boldsymbol{Y'})\to \\
(\underline{\boldsymbol{X'}}\multimap\underline{\boldsymbol{X}})\to(\boldsymbol{Y}\multimap\underline{\boldsymbol{Y'}})\to\mathsf{A}\to\mathsf{A}(\boldsymbol{X'},\boldsymbol{Y'},\underline{\boldsymbol{X'}},\underline{\boldsymbol{Y'}})
\end{aligned}
$$

The term M_{A} is defined by structural induction over A simultaneously with the definition of a term

$$
\begin{aligned}
N_{\underline{\mathsf{B}}}\colon \forall \boldsymbol{X},\boldsymbol{X'},\boldsymbol{Y},\boldsymbol{Y'},\underline{\boldsymbol{X}},\underline{\boldsymbol{X'}},\underline{\boldsymbol{Y}},\underline{\boldsymbol{Y'}}.\,(\boldsymbol{X'}\to\boldsymbol{X})\to(\boldsymbol{Y}\to\boldsymbol{Y'})\to \\
(\underline{\boldsymbol{X'}}\multimap\underline{\boldsymbol{X}})\to(\boldsymbol{Y}\multimap\underline{\boldsymbol{Y'}})\to\underline{\mathsf{B}}\multimap\underline{\mathsf{B}}(\boldsymbol{X'},\boldsymbol{Y'},\underline{\boldsymbol{X'}},\underline{\boldsymbol{Y'}})
\end{aligned}
$$

for any computation type $\underline{\mathsf{B}}$ satisfying the same condition on the occurrences of variables as above. The definition is in Figure 9, in which the simplified notation $\mathsf{A}(\boldsymbol{f},\boldsymbol{g},\boldsymbol{h},\boldsymbol{k})$ is used for M_{A} (or N_{A} whenever A is a computation type) applied to $\boldsymbol{f},\boldsymbol{g},\boldsymbol{h},\boldsymbol{k}$, alongside evident notation for function composition.

Lemma 10. *For computation types $\underline{\mathsf{A}}$ the terms $M_{\underline{\mathsf{A}}}$ and $N_{\underline{\mathsf{A}}}$ agree up to inclusion of $\multimap$ into $\to$. Moreover, the terms M_{A} define functors since they:*

- *preserve identities:* $\mathsf{A}(\boldsymbol{id},\boldsymbol{id},\boldsymbol{id},\boldsymbol{id}) = id$
- *preserve compositions:* $\mathsf{A}(\boldsymbol{f}\circ\boldsymbol{f'},\boldsymbol{g'}\circ\boldsymbol{g},\boldsymbol{h}\circ\boldsymbol{h'},\boldsymbol{k'}\circ\boldsymbol{k}) = \mathsf{A}(\boldsymbol{f'},\boldsymbol{g'},\boldsymbol{h'},\boldsymbol{k'})\circ\mathsf{A}(\boldsymbol{f},\boldsymbol{g},\boldsymbol{h},\boldsymbol{k})$

Finally we adapt the graph lemma of [7] to our setting.

Lemma 11 (Graph Lemma). *If $\boldsymbol{f}\colon\mathsf{B'}\to\mathsf{B},\boldsymbol{g}\colon\mathsf{C'}\to\mathsf{C},\boldsymbol{h}\colon\underline{\mathsf{B}}'\multimap\underline{\mathsf{B}},k\colon\underline{\mathsf{C}}\multimap\underline{\mathsf{C}}'$ then* L+P *proves*

$$\mathsf{A}[\langle\boldsymbol{f}\rangle^{\mathrm{op}},\langle\boldsymbol{g}\rangle,\langle\boldsymbol{h}\rangle^{\mathrm{op}},\langle\boldsymbol{k}\rangle]\equiv\langle\mathsf{A}(\boldsymbol{f},\boldsymbol{g},\boldsymbol{h},\boldsymbol{k})\rangle$$

Suppose $\underline{\mathsf{A}}$ is a computation type whose only free type variable is the computation type variable $\underline{X}$ which occurs only positively. As a consequence of parametric polymorphism the types

$$
\begin{aligned}
\mu^{\circ}\underline{X}.\,\underline{\mathsf{A}} &=_{\mathrm{def}} \forall\underline{X}.\,(\underline{\mathsf{A}}\multimap\underline{X})\to\underline{X} \\
\nu^{\circ}\underline{X}.\,\underline{\mathsf{A}} &=_{\mathrm{def}} \exists^{\circ}\underline{X}.\,(\underline{X}\multimap\underline{\mathsf{A}})\cdot\underline{X}
\end{aligned}
$$

are carriers of initial algebras and final coalgebras respectively for the functor induced by $\underline{\mathsf{A}}$. Here, we show how the universal property for final coalgebras can be verified in our logic.

The final coalgebra structure is defined using

$$\begin{aligned}\text{unfold}\colon \forall \underline{X}.\, (\underline{X} \multimap \underline{\mathsf{A}}) \to \underline{X} \multimap \nu^{\circ} \underline{X}.\, \underline{\mathsf{A}} \\ \text{out}\colon (\nu^{\circ} \underline{X}.\, \underline{\mathsf{A}}) \multimap \underline{\mathsf{A}}[\nu^{\circ} \underline{X}.\, \underline{\mathsf{A}}/\underline{X}]\end{aligned}$$

defined as

$$\begin{aligned}\text{unfold} &= \Lambda \underline{X}.\, \lambda f\colon \underline{X} \multimap \underline{\mathsf{A}}.\, \lambda^{\circ} x\colon (\nu^{\circ} \underline{X}.\, \underline{\mathsf{A}}).\, \langle \underline{X}, f \cdot x \rangle \\ \text{out} &= \lambda^{\circ} x\colon \nu^{\circ} \underline{X}.\, \underline{\mathsf{A}}.\, \text{let } \langle \underline{X}, y \rangle \text{ be } x \text{ in } (\text{let } f \cdot z \text{ be } y \text{ in } \underline{\mathsf{A}}(\text{unfold } \underline{X}\ f)(f(z)))\end{aligned}$$

Lemma 12. *If* $\Gamma \mid - \vdash f\colon \underline{\mathsf{B}} \multimap \underline{\mathsf{A}}[\underline{\mathsf{B}}/\underline{X}]$ *then* L *proves*

$$\Gamma \mid x\colon \underline{\mathsf{B}} \vdash \text{out}(\text{unfold } \underline{\mathsf{B}}\ f\ x) = \underline{\mathsf{A}}(\text{unfold } \underline{\mathsf{B}}\ f)(f(x))\,.$$

In diagramatic form Lemma 12 means that

$$\begin{array}{ccc} \underline{\mathsf{B}} & \xrightarrow{\quad f \quad} & \underline{\mathsf{A}}[\underline{\mathsf{B}}/\underline{X}] \\ {\scriptstyle \text{unfold } \underline{\mathsf{B}}\ f}\big\downarrow & & \big\downarrow{\scriptstyle A(\text{unfold } \underline{\mathsf{B}}\ f)} \\ \nu^{\circ} \underline{X}.\, \underline{\mathsf{A}} & \xrightarrow{\quad \text{out} \quad} & \underline{\mathsf{A}}[\nu^{\circ} \underline{X}.\, \underline{\mathsf{A}}/\underline{X}] \end{array}$$

commutes, i.e., the term unfold verifies that out is a weak final coalgebra.

Lemma 13. *Suppose*

$$\Gamma \mid - \vdash h\colon \underline{\mathsf{B}} \multimap \underline{\mathsf{B}}',\ f\colon \underline{\mathsf{B}} \multimap \underline{\mathsf{A}}[\underline{\mathsf{B}}/\underline{X}],\ f\colon \underline{\mathsf{B}}' \multimap \underline{\mathsf{A}}[\underline{\mathsf{B}}'/\underline{X}],$$

and that $\Gamma \mid x\colon \underline{\mathsf{B}} \vdash f'(h(x)) = \underline{\mathsf{A}}(h)(f(x))$. *Then* L+P *proves*

$$\Gamma \mid x\colon \underline{\mathsf{B}} \vdash \text{unfold } \underline{\mathsf{B}}\ f\ x = \text{unfold } \underline{\mathsf{B}}'\ f'\ (h(x))$$

Proof. By the Graph Lemma (Lemma 11) the assumption can be reformulated as $\langle h \rangle \multimap \underline{\mathsf{A}}[\langle h \rangle/\underline{X}])(f, f')$. So by parametricity of unfold

$$(\langle h \rangle \multimap eq_{\nu^{\circ} \underline{X}.\, \underline{\mathsf{A}}})(\text{unfold } \underline{\mathsf{B}}\ f, \text{unfold } \underline{\mathsf{B}}'\ f')$$

implying $\Gamma \mid x\colon \underline{\mathsf{B}} \vdash \text{unfold } \underline{\mathsf{B}}\ f\ x = \text{unfold } \underline{\mathsf{B}}'\ f'\ (h(x))$. □

Lemma 14. L+P *proves* $\text{unfold } \nu^{\circ} \underline{X}.\, \underline{\mathsf{A}}\, \text{out} = id_{\nu^{\circ} \underline{X}.\, \underline{\mathsf{A}}}$.

Proof. By Lemma 13, for arbitrary $\underline{X}, f\colon \underline{X} \multimap \underline{\mathsf{A}}$,

$$\text{unfold } \underline{X}\ f = (\text{unfold } \nu^{\circ} \underline{X}.\, \underline{\mathsf{A}}\, \text{out}) \circ (\text{unfold } \underline{X}\ f)$$

so $\Gamma, f\colon \underline{X} \multimap \underline{\mathsf{A}} \mid x\colon \underline{X} \vdash \langle \underline{X}, f \cdot x\rangle = \mathrm{unfold}\, \nu^\circ \underline{X}.\, \underline{\mathsf{A}}\, \mathrm{out}\, \langle \underline{X}, x\rangle$. This implies using Lemma 8 and Lemma 9 that for any $y\colon \nu^\circ \underline{X}.\, \underline{\mathsf{A}}$

$$\begin{aligned} y &= \mathrm{let}\ \langle \underline{X}, f \cdot x\rangle\ \mathrm{be}\, y\ \mathrm{in}\ \langle \underline{X}, f \cdot x\rangle \\ &= \mathrm{let}\ \langle \underline{X}, f \cdot x\rangle\ \mathrm{be}\, y\ \mathrm{in}\ (\mathrm{unfold}\, \nu^\circ \underline{X}.\, \underline{\mathsf{A}}\, \mathrm{out}\, \langle \underline{X}, x\rangle) \\ &= \mathrm{unfold}\, \nu^\circ \underline{X}.\, \underline{\mathsf{A}}\, \mathrm{out}\, (\mathrm{let}\ \langle \underline{X}, f \cdot x\rangle\ \mathrm{be}\, y\ \mathrm{in}\ \langle \underline{X}, x\rangle) \\ &= \mathrm{unfold}\, \nu^\circ \underline{X}.\, \underline{\mathsf{A}}\, \mathrm{out}\, y \end{aligned}$$

and so the lemma follows from extensionality. □

Theorem 4. *Suppose* $f\colon \underline{\mathsf{B}} \multimap \nu^\circ \underline{X}.\underline{\mathsf{A}}$ *and* $h\colon \underline{\mathsf{B}} \multimap \underline{\mathsf{A}}[\underline{\mathsf{B}}/\underline{X}]$ *such that* $\underline{\mathsf{A}}(h) \circ f = \mathrm{out} \circ h$ *then* L+P *proves* $h = \mathrm{unfold}\, \underline{\mathsf{B}}\, f$.

Proof. By Lemma 13 and Lemma 14

$$\mathrm{unfold}\, \underline{\mathsf{B}}\, f = (\mathrm{unfold}\, \nu^\circ \underline{X}.\, \underline{\mathsf{A}}\, \mathrm{out}) \circ h = h\,.$$

□

References

1. 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.
2. L. Birkedal, R. E. Møgelberg, and R. L. Petersen. Linear Abadi & Plotkin logic. *Logical Methods in Computer Science*, 2, 2006.
3. P. B. Levy. *Call By Push Value, a Functional/Imperative Synthesis.* Kluwer, 2004.
4. R. E. Møgelberg and A. K. Simpson. Relational parametricity for computational effects. In *LICS*, pages 346–355. IEEE Computer Society, 2007.
5. R. E. Møgelberg and A. K. Simpson. Relational parametricity for control considered as a computational effect. *Electr. Notes Theor. Comput. Sci*, 173:295–312, 2007.
6. E. Moggi. Notions of computation and monads. *Information and Computation*, 93:55–92, 1991.
7. G. D. Plotkin and M. 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.
8. G. D. Plotkin and J. Power. Algebraic operations and generic effects. *Applied Categorical Structures*, 11(1):69–94, 2003.
9. 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.
10. J. C. Reynolds. Types , abstraction, and parametric polymorphism. *Information Processing*, 83:513–523, 1983.
11. C. Strachey. Fundamental concepts in programming languages. Lecture Notes, International Summer School in Computer Programming, Copenhagen, August 1967.
12. P. Wadler. Theorems for free. In *Proceedings 4th International Conference on Functional Programming languages and Computer Architectures*, 1989.