Relational Parametricity for Computational Effects

Rasmus Ejlers Møgelberg* Alex Simpson†
LFCS, School of Informatics
University of Edinburgh, Scotland, UK

Abstract

According to Strachey, a polymorphic program is parametric if it applies a uniform algorithm independently of the type instantiations at which it is applied. The notion of relational parametricity, introduced by Reynolds, is one possible mathematical formulation of this idea. Relational parametricity provides a powerful tool for establishing data abstraction properties, proving equivalences of datatypes, and establishing equalities of programs. Such properties have been well studied in a pure functional setting. Real programs, however, exhibit computational effects. In this paper, we develop a framework for extending the notion of relational parametricity to languages with effects.

1. Introduction

The theory of *relational parametricity*, proposed by Reynolds [21], provides a powerful framework for establishing properties of polymorphic programs and their types. Such properties include the "theorems for free" of Wadler [26], universal properties for datatype encodings, and representation independence properties for abstract datatypes. These results are well established, see e.g. [18], for the pure Girard/Reynolds second-order λ-calculus (a.k.a. system F) which provides a concise yet remarkably powerful calculus of typed total functions.

The generalisation of relational parametricity to richer calculi can be problematic. Even the addition of recursion (hence nontermination) causes difficulties, since the fixed-point property of recursion is incompatible with certain consequences of relational parametricity as usually formulated.[1] This issue led Plotkin [17] to propose using second-order linear type theory as a framework for combining parametricity and recursion. Such an approach has been further investigated in [1, 2]. One of its many good properties is that it supports a rich collection of polymorphic datatype encodings with the desired universal properties following from relational parametricity.

The addition of recursion is just one possible extension of second-order λ-calculus. For example, in [14], Parigot (implicitly) considers an othogonal extension obtained by adding control operators. Recently, M. Hasegawa [5] has developed a syntactic account of relational parametricity for Parigot's calculus. An intriguing fact he observes is that, even though the technical frameworks for the two approaches are quite different, there are striking analogies between his "focal" parametricity and Plotkin's linear parametricity. Accordingly, Hasegawa poses the question of whether it is possible to find a unifying framework for relational parametricity that includes both his work and Plotkin's linear parametricity as special cases.

In this paper we answer this question by providing a general theory of relational parametricity for computational effects. Not only does our approach generalise both Plotkin's and Hasegawa's, but it also applies across the full range of computational effects (e.g., nondeterminism, probabilistic choice, input/output, side effects, exceptions, etc.).

We build on the work of Moggi [12, 13], who proposed incorporating effects into type theory by adding a new type constructor for typing "computations" rather than values. For every type B, one has a new type $!\,\mathsf{B}$ (our non-standard notation is justified in Section 5) whose elements represent computations that (potentially) return values in B, and which (possibly) perform effects along the way. Semantically, $!$ is interpreted using a *computational monad* that encapsulates the relevant kinds of effect.

In order to obtain an account of relational parametricity for monads, one needs to solve a problem. Basic to relational parametricity is the idea of treating types as relations. Polymorphic functions are required to preserve derived relations under all possible instantiations of relations to type variables. To extend this to computational effects it is necessary to determine how the operation $!$ determines a relation $!\,R \subseteq\, !\,\mathsf{A} \times !\,\mathsf{B}$ from any relation $R \subseteq \mathsf{A} \times \mathsf{B}$. That is one needs a "relational lifting" of the $!$ operation. The literature

*Research supported by EPSRC and the Danish Agency for Science, Technology and Innovation.
†Research supported by an EPSRC Advanced Research Fellowship.
[1] Relational parametricity implies types form a cartesian closed category with finite sums, and any such category with fixed points is trivial.

contains two approaches to defining such a relational lifting for ! [4, 8] (although neither is presented in the context of polymorphism). In the present paper we instead side-step the issue in a surprising way: we show that, given the right choice of underlying type theory, ! is polymorphically definable in terms of more basic primitives.

Our type theory, which we call PE, is presented in Section 2. It is closely related to Levy's system of *call by push-value (CBPV)* [9], which subsumes call-by-name and call-by-value calculi with effects. Levy emphasises the importance of having two general classes of type: *value types*, which classify "values", and *computation types*, which classify "computations". The intuitive difference between the two is that "a value *is*" and "a computation *does*". Technically, this intuition is supported by a wealth of semantic and operational interpretations of the framework, see [9].

With general computation types at hand, one can give the ! constructor the following polymorphic definition:

$$!\,\mathsf{B} \;=_{\text{def}}\; \forall \underline{X}.\, (\mathsf{B} \to \underline{X}) \to \underline{X} \quad (\underline{X} \text{ not free in } \mathsf{B}), \quad (1)$$

where importantly the type variable $\underline{X}$ ranges over computation types only. As we shall see, the type constructors used in the definition all have natural relational interpretations, and hence the defined ! operation inherits an induced relational lifting.

In order to reason about parametricity in PE, we build a relationally parametric model of our calculus. Even in the case of ordinary second-order λ-calculus, the construction of parametric models is non-trivial. In our case, the interaction between value and computation types contributes significant additional complexities. To keep things as simple as possible, we work with a set-theoretic model, exploiting the fact that it is consistent to do so if one keeps to intuitionistic reasoning. The details are presented in Sections 3 and 4. As a first application of the model, we prove in Section 5 that the ! operator, as defined by (1) above, does indeed enjoy its expected universal property (Theorem 5.2).

In Section 6, we consider how to specialise the generic calculus PE to specific effects of interest. One useful form of specialisation recurs in many examples. It is common for effects to have associated operations that trigger and/or react to "effectful" behaviour. Typically, one would like to give an n-ary such operation the polymorphic type:

$$\forall X.\;\; (!X)^n \;\to\; !X \;\; . \quad (2)$$

For example, a binary nondeterministic choice operation forms a computation by choosing between two possible continuation computations. Also, the "handle" operation for an exception e, can be viewed as a binary operation where $\mathrm{handle}^e(p, q)$ behaves like p unless p raises exception e, in which case q is executed. Since such operations are computed in a type-independent way, they are "parametric" in the informal sense of Strachey. We show that such operations are also parametric according to our theory of relational parametricity. This involves two technical developments of independent interest. The first relates to recent work by Plotkin and Power [20], in which they observe that many operations on effects are "algebraic operations" in the sense of universal algebra. As Theorem 6.1, we obtain that n-ary algebraic operations are in one-to-one correspondence with (parametric) elements of type:

$$\forall \underline{X}.\; \underline{X}^n \;\to\; \underline{X} \;\; , \quad (3)$$

where again $\underline{X}$ ranges over computation types. Thus algebraic operations can be incorporated within PE as constants of the above type (which is more informative than (2), since monadic types $!\,\mathsf{B}$ are always computation types).

Not all useful operations on effects arise as algebraic operations; e.g., exception handling is a counterexample. However, exception handling can be added to PE using a different strengthening of (2) for its type:

$$\forall X.\;\; (!X)^2 \;\multimap\; !X \;\; . \quad (4)$$

This correctness of this typing is again based on a general result (Theorem 6.2) which characterises the (parametric) elements of the above type (the nature of the linear arrow is explained in the sequel) in terms of a naturality condition.

Finally, in Section 7, we outline the relationship between PE and other approaches to parametricity and effects. Plotkin's linear parametricity arises as a specialisation of PE valid in the special case of "commutative" monads. We also briefly discuss how Hasegawa's account of parametricity and control arises as a specialisation of PE. The details for this appear in a companion paper [10].

Acknowledgements We are indebted to Masahito Hasegawa for first suggesting that (1) should be a general phenomenon within a monadic framework incorporating both linear and continuation-passing settings as special cases. We thank him and Paul Levy for helpful discussions.

2. A polymorphic calculus

We start by defining the type theory PE for polymorphism and effects. As discussed in the introduction, following [9], PE contains both *value types* $\mathsf{A}, \mathsf{B}, \mathsf{C}, \ldots$ and *computation types* $\underline{\mathsf{A}}, \underline{\mathsf{B}}, \underline{\mathsf{C}}, \ldots$. A central feature of our type theory is that we allow polymorphic type quantification over both value types and computation types. Accordingly, we use $X, Y, Z, \ldots$ to range over a countable set of value-type variables, and $\underline{X}, \underline{Y}, \underline{Z}, \ldots$ to range over a disjoint countable set of computation-type variables. Value types and computation types are then mutually defined by:

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

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

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

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

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

Figure 1. Typing rules.

Note that the computation types form a subcollection of the value types. The intuition here is that any (active) computation has a corresponding (static) value, its "thunk". In contrast to [9], we make this passage from computations to values syntactically invisible.

For semantic intuition, one should think of value types as representing sets, and of computation types as representing Eilenberg-Moore algebras for some computational monad on sets. Then $\mathsf{B} \to \mathsf{C}$ is the set of all functions. The special case $\mathsf{B} \to \underline{\mathsf{A}}$ is a computation type because algebras are closed under powers, with the algebra structure defined pointwise. The type $\underline{\mathsf{A}} \multimap \underline{\mathsf{B}}$ represents the set of all algebra homomorphisms from $\underline{\mathsf{A}}$ to $\underline{\mathsf{B}}$. In general, there is no natural algebra structure on this set, hence the type $\underline{\mathsf{A}} \multimap \underline{\mathsf{B}}$ is not a computation type. Finally $\forall X.\, \mathsf{B}$ and $\forall \underline{X}.\, \mathsf{B}$ are polymorphic types, with the polymorphism ranging over value types and computation types respectively. In either case, when B is a computation type, the polymorphic type is again a computation type. This is justified by Proposition 4.1 below.

Our types, which are based on function spaces and polymorphism, are are not directly comparable with Levy's [9], which include sums and products. Nonetheless, we shall see in Section 7 that we can encode Levy's calculus within ours. Given this, our calculus extends Levy's with polymorphic types (cf. [9, §12.4]) and linear function types. In fact, the latter have a particularly nice explanation in terms of Levy's stack-based operational framework, within which a value of type $\underline{\mathsf{A}} \multimap \underline{\mathsf{B}}$ can be understood as a stack turning a computation of type $\underline{\mathsf{A}}$ into a computation of type $\underline{\mathsf{B}}$.

Having computation types as special value types allows us to base our type system on a single judgement form:

$$\Gamma \,|\, \Delta \,\vdash\, t\colon \mathsf{B} \ ,$$

where Γ and Δ are disjoint contexts of variable typings subject to the following conditions: either (i) Δ is empty, or (ii) B is a computation type and Δ has the form $x\colon \underline{\mathsf{A}}$, where $\underline{\mathsf{A}}$ is also a computation type. Thus the context Δ, which, following [3], we call the *stoup* of the typing judgement, contains at most one typing assertion. When we want to be explicit about which of (i) or (ii) applies, we write:

$$\begin{array}{ll} \text{(i)} & \Gamma \,|\, - \,\vdash\, t\colon \mathsf{B} \\ \text{(ii)} & \Gamma \,|\, x\colon \underline{\mathsf{A}} \,\vdash\, t\colon \underline{\mathsf{B}} \ . \end{array}$$

In the first case, the intuitive interpretation of t is as an arbitrary function from the product of all types in Γ to the type B. In the second case, the interpretation of t is as a function from $\Gamma \times \underline{\mathsf{A}}$ to $\underline{\mathsf{B}}$ that is an algebra homomorphism in its right-hand argument (i.e. for every fixed set of values for the Γ variables, the induced function from $\underline{\mathsf{A}}$ to $\underline{\mathsf{B}}$ is a homomorphism). From this interpretation, one sees why the stoup is restricted to computation types, and also why, when the stoup is nonempty, the result type is required to be a computation type. (Similar considerations are in fact familiar from other stoup-based calculi, e.g., Girard's LU [3].)

The type system is presented in Figure 1. The side conditions refer to the set $\mathrm{ftv}(\Gamma)$ of free type variables in a context Γ, which is defined in the obvious way. Of course, the type rules are restricted to apply only when the premises satisfy the conditions on judgements imposed above. In such cases, the rule conclusions also satisfy these conditions.

It is immediate that the type system for value types extends the standard second-order λ-calculus of Girard and Reynolds. Indeed, the typing rules for the relevant types (X, $\mathsf{B} \to \mathsf{C}$ and $\forall X.\, \mathsf{B}$), when restricted to the case with empty stoup, are just the usual ones. It is well-known that the second-order λ-calculus is powerful enough to encode many type constructors including products, sums, inductive and coinductive types. We include those definitions we shall need later in Figure 2.

$$1 =_{\mathrm{def}} \forall X.\, X \to X$$
$$\mathsf{A} \times \mathsf{B} =_{\mathrm{def}} \forall X.\, (\mathsf{A} \to \mathsf{B} \to X) \to X \qquad (X \notin \mathrm{ftv}(\mathsf{A},\mathsf{B}))$$
$$0 =_{\mathrm{def}} \forall X.\, X$$
$$\mathsf{A} + \mathsf{B} =_{\mathrm{def}} \forall X.\, (\mathsf{A} \to X) \to (\mathsf{B} \to X) \to X \ (X \notin \mathrm{ftv}(\mathsf{A},\mathsf{B}))$$

Figure 2. Definable value types

3. Semantic setting

In the previous section, we appealed to semantic intuition by explaining value types as sets and computation types as algebras for a monad on sets. Unfortunately, this intuition runs into the technical problem that there are no set-theoretic models of polymorphism [22]. However, it was shown by Pitts [15] that set-theoretic models of polymorphism are possible if *intuitionistic* set theory is used rather than ordinary *classical* set theory. We shall exploit this by working with such an intuitionistic set-theoretic model. The advantage of this strategy is that the set-theoretic framework allows the development to concentrate entirely on the difficulties inherent in defining a suitable notion of relational parametricity, which are formidable in themselves, rather than on incidental details specific to a particular concrete model. Our approach results in no loss of generality. All denotational models of relational parametricity of which we are aware can be exhibited as full subcategories of models of intuitionistic set theory.

Henceforth in this paper, we use Friedman's Intuitionistic Zermelo-Fraenkel set theory (IZF) as our meta-theory, see e.g. [25]. IZF is the established intuitionistic counterpart of classical Zermelo-Fraenkel set theory (ZF). Just as in ordinary mathematics one works informally in ZF, we shall work similarly informally within IZF. Readers who are not familiar with IZF and the distinctions between intuitionistic and classical reasoning will anyway be able to follow the development, since IZF is a subtheory of ZF. However, such readers will have to place their trust in the authors that the reasoning principles of IZF are never violated.

Value types will be modelled as sets, but it is known that it is not possible to interpret types in the second-order λ-calculus as arbitrary sets [16]. Thus we require a collection of special sets for interpreting types. Such special sets need to be closed under the set-theoretic operations used in the interpretation. Accordingly, we assume that we have a full subcategory $\mathcal{C}$ of the category $\mathbf{Set}$ of sets that satisfies:

(C1) If $A \in \mathcal{C}$ and $A \cong B$ in $\mathbf{Set}$ then $B \in \mathcal{C}$.

(C2) For any set-indexed family $\{A_i\}_{i \in I}$ of sets in $\mathcal{C}$, the set-theoretic product $\prod_{i \in I} A_i$ is again in $\mathcal{C}$.

(C3) Given $A, B \in \mathcal{C}$ and functions $f, g \colon A \to B$, the equalizer $\{x \in A \mid f(x) = g(x)\}$ is again in $\mathcal{C}$.

(C4) There is a set $\mathbf{C}$ of objects of $\mathcal{C}$ such that, for any $A \in \mathcal{C}$, there exists $B \in \mathbf{C}$ with $B \cong A$.

By items (C2) and (C3), the category $\mathcal{C}$ is small-complete with limits inherited from $\mathbf{Set}$. Since function spaces are powers, for any set A and any $B \in \mathcal{C}$, the function space B^A is in $\mathcal{C}$, i.e. $\mathcal{C}$ is an *exponential ideal* of $\mathbf{Set}$. In particular, $\mathcal{C}$ is cartesian closed. By (C1), the category $\mathcal{C}$ is not a small category. However, by (C4) it is weakly equivalent to its small full subcategory on the set of objects $\mathbf{C}$.

In classical set theory, the above conditions imply that every object in $\mathcal{C}$ is either the empty set or a singleton set. The reason we work in IZF is that this renders it consistent for $\mathcal{C}$ to be an interesting category. Indeed, it is consistent for the natural numbers to be an object of $\mathcal{C}$. This consistency property derives from the work of Hyland *et. al.* on small-complete small categories [6, 7]. However, our perspective is different. Rather than assuming a small category that is complete only in a restricted technical sense [7, 23], our category $\mathcal{C}$ is genuinely complete, but only weakly equivalent to a small category. This approach, which simplifies the development, is taken from [24].

According to our informal explanation of computation types in Section 2, they should be interpreted as Eilenberg-Moore algebras for a monad T on $\mathcal{C}$. For any such monad T, the category $\mathcal{A}$ of algebras comes with a forgetful functor $U \colon \mathcal{A} \to \mathcal{C}$ and the following properties are satisfied.

(A1) U "weakly creates limits" in the following sense. For every diagram Δ in $\mathcal{A}$ and limiting cone $\lim U(\Delta)$ of $U(\Delta)$ in $\mathcal{C}$, there exists a specified limiting cone $\lim \Delta$ of Δ in $\mathcal{A}$ such that $U(\lim \Delta) = \lim U(\Delta)$.

(A2) U reflects isomorphisms (i.e. if Uf is an isomorphism in $\mathcal{C}$ then f is an isomorphism in $\mathcal{A}$).

(A3) For objects $\underline{A}, \underline{B}$ of $\mathcal{A}$, the hom-set $\mathcal{A}(\underline{A}, \underline{B})$ is an object of $\mathcal{C}$.

(A4) There exists a set $\mathbf{A}$ of objects of $\mathcal{A}$ such that for every $\underline{A} \in \mathcal{A}$, there exists $\underline{B} \in \mathbf{A}$ with $\underline{B}$ isomorphic to $\underline{A}$.

Indeed, (A1) and (A2) are standard. Property (A3) holds because $\mathcal{A}(\underline{A}, \underline{B})$ arises as an equalizer in $\mathcal{C}$ of two evident functions $(U\underline{B})^{U\underline{A}} \to (U\underline{B})^{TU\underline{A}}$. Also, (A4) holds because the collection of algebra structures on objects of $\mathbf{C}$ is a set.

The reason for identifying (A1)–(A4) is that, in order to interpret the calculus of Section 2, it is sufficient to work with any category $\mathcal{A}$ and functor $U \colon \mathcal{A} \to \mathcal{C}$ satisfying (A1)–(A4) above.[2] Henceforth, we assume this situation.

It is convenient to maintain algebraic terminology for the category $\mathcal{A}$. Thus we call the objects of $\mathcal{A}$ *algebras*. By (A1) and (A2), the functor U is faithful, thus we can identify the morphisms $\mathcal{A}(\underline{A}, \underline{B})$ with special functions from $U\underline{A}$ to $U\underline{B}$, which we call *homomorphisms*. We write $\underline{A} \multimap \underline{B}$ for

[2] In particular, the weakening of limit creation in (A1) is crucial to the application in [10].

the set of homomorphisms from $\underline{A}$ to $\underline{B}$. (N.B. by (A3) the set $\underline{A} \multimap \underline{B}$ is an object of $\mathcal{C}$.)

In Section 4 we interpret the type theory of Section 2 using $U\colon \mathcal{A} \to \mathcal{C}$. In doing so, we formulate relational parametricity using binary relations in $\mathcal{C}$ and $\mathcal{A}$. As usual, these are defined as subobjects of products. First, we review the basic properties of subobjects in $\mathcal{C}$ and $\mathcal{A}$.

We write $\mathrm{Sub}_{\mathcal{C}}(A)$ for the set of subobjects of A in $\mathcal{C}$. Since the inclusion $\mathcal{C} \hookrightarrow \mathbf{Set}$ preserves limits and hence monomorphisms, this is explicitly defined by:

$$\mathrm{Sub}_{\mathcal{C}}(A) \;=\; \{B \in \mathcal{C} \mid B \subseteq A\}\,.$$

We call the elements of $\mathrm{Sub}_{\mathcal{C}}(A)$ the *$\mathcal{C}$-subsets* of A.

Similarly, we write $\mathrm{Sub}_{\mathcal{A}}(\underline{A})$ for the collection of subobjects of an algebra $\underline{A}$ in $\mathcal{A}$. Because U preserves limits, every mono $\underline{B} \rightarrowtail \underline{A}$ in $\mathcal{A}$ is mapped by U to a mono $U\underline{B} \rightarrowtail U\underline{A}$ in $\mathcal{C}$. Thus, for every $\underline{A} \in \mathcal{A}$, the functor U determines a function $\mathrm{Sub}_{\mathcal{A}}(\underline{A}) \to \mathrm{Sub}_{\mathcal{C}}(U\underline{A})$. By (A1) and (A2), this function preserves and reflects the ordering. We say that $A \subseteq U\underline{A}$ *carries a subalgebra* if it represents a subobject in the image of the map $\mathrm{Sub}_{\mathcal{A}}(\underline{A}) \to \mathrm{Sub}_{\mathcal{C}}(U\underline{A})$ induced by U. In fact, $\mathrm{Sub}_{\mathcal{A}}(\underline{A})$ is given explicitly by:

$$\mathrm{Sub}_{\mathcal{A}}(\underline{A}) = \{B \in \mathcal{C} \,|\, B \subseteq U\underline{A} \text{ and carries a subalg. of } \underline{A}\}\ .$$

We introduce notation for binary relations. For $A \in \mathcal{C}$, we write Δ_A for the diagonal (identity) relation in $\mathrm{Sub}_{\mathcal{C}}(A \times A)$. Similarly, for $\underline{A} \in \mathcal{A}$, we write $\Delta_{\underline{A}}$ for the diagonal relation on $U\underline{A}$, which is indeed in $\mathrm{Sub}_{\mathcal{A}}(\underline{A} \times \underline{A})$. For $f\colon A' \to A, g\colon B' \to B$ in $\mathcal{C}$ and $R \in \mathrm{Sub}_{\mathcal{C}}(A \times B)$ we write $(f,g)^{-1}R$ for $\{(x,y) \mid (f(x), g(y)) \in R\}$. Notice that if $f\colon \underline{A}' \multimap \underline{A}, g\colon \underline{B}' \multimap \underline{B}$ in $\mathcal{A}$ and $Q \in \mathrm{Sub}_{\mathcal{A}}(\underline{A} \times \underline{B})$ then $(f,g)^{-1}Q \in \mathrm{Sub}_{\mathcal{A}}(\underline{A}' \times \underline{B}')$.

To formulate relational parametricity, we require two specified collections of *admissible* relations, one $\mathcal{R}_{\mathcal{C}}(A,B) \subseteq \mathrm{Sub}_{\mathcal{C}}(A \times B)$ on objects of $\mathcal{C}$ and one $\mathcal{R}_{\mathcal{A}}(\underline{A},\underline{B}) \subseteq \mathrm{Sub}_{\mathcal{A}}(\underline{A} \times \underline{B})$ on objects of $\mathcal{A}$. These are required to satisfy:

(R1) For each object A of $\mathcal{C}$ the diagonal relation Δ_A is in $\mathcal{R}_{\mathcal{C}}(A,A)$ and likewise for each object $\underline{A}$ of $\mathcal{A}$ the diagonal $\Delta_{\underline{A}}$ is in $\mathcal{R}_{\mathcal{A}}(\underline{A},\underline{A})$.

(R2) Admissible relations are closed under reindexing, i.e., if $R \in \mathcal{R}_{\mathcal{C}}(A,B)$ and $f\colon A' \to A$, $g\colon B' \to B$, then $(f,g)^{-1}R \in \mathcal{R}_{\mathcal{C}}(A',B')$ and if $Q \in \mathcal{R}_{\mathcal{A}}(\underline{A},\underline{B})$ and $f\colon \underline{A}' \multimap \underline{A}$, $g\colon \underline{B}' \multimap \underline{B}$, then $(f,g)^{-1}Q \in \mathcal{R}_{\mathcal{A}}(\underline{A}',\underline{B}')$

(R3) For any set of admissible $\mathcal{C}$- (respectively $\mathcal{A}$-)relations on the same pair of objects, the intersection is an admissible $\mathcal{C}$- (respectively $\mathcal{A}$-)relation.

(R4) $\mathcal{R}_{\mathcal{A}}(\underline{A},\underline{B}) \subseteq \mathcal{R}_{\mathcal{C}}(U\underline{A},U\underline{B})$.

(R1) and (R2) imply that graphs of functions are admissible, i.e., if $f\colon A \to B$ then $\langle f\rangle =_{\mathrm{def}} \{(x,y) \mid f(x) = y\} \in \mathcal{R}_{\mathcal{C}}(A,B)$ and if $g\colon \underline{A} \multimap \underline{B}$ then $\langle g\rangle \in \mathcal{R}_{\mathcal{A}}(\underline{A},\underline{B})$.

By a *parametric model of PE* we shall mean any category $\mathcal{C}$ satisfying (C1)–(C4), together with a category $\mathcal{A}$ and functor $U\colon \mathcal{A} \to \mathcal{C}$ satisfying (A1)–(A4) and collections $\mathcal{R}_{\mathcal{C}}$ and $\mathcal{R}_{\mathcal{A}}$ satisfying (R1)–(R4) above. The proposition below shows that every monad gives rise to a parametric model of PE. Thus the theory of relational parametricity for PE that we shall develop over such models is applicable to arbitrary computational monads.

Proposition 3.1 *Given $\mathcal{C}$ satisfying (C1)–(C4) and a monad T on $\mathcal{C}$, let $\mathcal{A}$ be the category of algebras for the monad, U the forgetful functor and define $\mathcal{R}_{\mathcal{C}}(A,B) = \mathrm{Sub}_{\mathcal{C}}(A \times B)$ and $\mathcal{R}_{\mathcal{A}}(\underline{A},\underline{B}) = \mathrm{Sub}_{\mathcal{A}}(\underline{A} \times \underline{B})$. This data defines a parametric model of PE.*

Although the above is a useful general result, we comment that some applications of PE require a different choice of model. For example, the application of PE to control in [10] makes crucial use of the permitted flexibility in the choice of $\mathcal{A}$, U and $\mathcal{R}_{\mathcal{A}}$. (The situation is analogous to that for Levy's CBPV [9], where the natural adjunction model of control does not involve the Eilenberg-Moore category.)

4. Interpreting the calculus

In this section we interpret PE in any parametric model as defined in Section 3. As adumbrated there, a value type B will be interpreted as a set $\mathcal{C}[\![\mathsf{B}]\!]$ in $\mathcal{C}$, and a computation type $\underline{\mathsf{A}}$ will be interpreted as an algebra $\mathcal{A}[\![\underline{\mathsf{A}}]\!]$. In order to incorporate relational parametricity, we shall also give a second interpretation of a value type B as an admissible $\mathcal{C}$-relation $\mathcal{R}[\![\mathsf{B}]\!]$. For computation types $\underline{\mathsf{A}}$, it will hold automatically that $\mathcal{R}[\![\underline{\mathsf{A}}]\!]$ is also an admissible $\mathcal{A}$-relation.

Given a set of type variables Θ, a Θ-*environment* is a function γ mapping every value-type variable $X \in \Theta$ to an object $\gamma(X)$ of $\mathcal{C}$, and every computation-type variable $\underline{X} \in \Theta$ to an object $\gamma(\underline{X})$ of $\mathcal{A}$. A *relational* Θ-*environment* is a tuple $\rho = (\rho_1, \rho_2, \rho_{\mathcal{R}})$, where: ρ_1, ρ_2 are Θ-environments; for every value-type variable $X \in \Theta$,

$$\rho_{\mathcal{R}}(X) \in \mathcal{R}_{\mathcal{C}}(\rho_1(X), \rho_2(X))\ \ ;$$

and, for every computation-type variable $\underline{X} \in \Theta$,

$$\rho_{\mathcal{R}}(\underline{X}) \in \mathcal{R}_{\mathcal{A}}(\rho_1(\underline{X}), \rho_2(\underline{X}))\ \ .$$

For each value type $\mathsf{B}(\Theta)$ (i.e. type B with $\mathrm{ftv}(\mathsf{B}) \subseteq \Theta$) and Θ-environment γ, we define an object $\mathcal{C}[\![\mathsf{B}]\!]_{\gamma}$ of $\mathcal{C}$; and, for each computation type $\underline{\mathsf{A}}(\Theta)$ and Θ-environment γ, we define an object $\mathcal{A}[\![\underline{\mathsf{A}}]\!]_{\gamma}$ of $\mathcal{A}$. Interdependently with the above, for each value type $\mathsf{B}(\Theta)$ and relational Θ-environment ρ, we define an admissible $\mathcal{C}$-relation $\mathcal{R}[\![\mathsf{B}]\!]_{\rho} \in \mathcal{R}_{\mathcal{C}}(\mathcal{C}[\![\mathsf{B}]\!]_{\rho_1}, \mathcal{C}[\![\mathsf{B}]\!]_{\rho_2})$. The definitions are given in Figure 3 (although see below for an important caveat). In these definitions, the products and powers used in the definition of

$$
\begin{aligned}
\mathcal{C}[\![X]\!]_\gamma &= \gamma(X)\\
\mathcal{C}[\![\mathsf{B} \to \mathsf{C}]\!]_\gamma &= \mathcal{C}[\![\mathsf{C}]\!]_\gamma{}^{\mathcal{C}[\![\mathsf{B}]\!]_\gamma}\\
\mathcal{C}[\![\forall X.\, \mathsf{B}]\!]_\gamma &= \{\pi \in \prod_{A \in \mathcal{C}} \mathcal{C}[\![\mathsf{B}]\!]_{\gamma[A/X]} \mid \forall A, B \in \mathcal{C},\, \forall R \in \mathcal{R}_{\mathcal{C}}(A,B).\ \mathcal{R}[\![\mathsf{B}]\!]_{\Delta_{\gamma}[R/X]}(\pi_A, \pi_B)\}\\
\mathcal{C}[\![\underline{X}]\!]_\gamma &= U(\gamma(\underline{X}))\\
\mathcal{C}[\![\underline{\mathsf{A}} \multimap \underline{\mathsf{B}}]\!]_\gamma &= \mathcal{A}[\![\underline{\mathsf{A}}]\!]_\gamma \multimap \mathcal{A}[\![\underline{\mathsf{B}}]\!]_\gamma\\
\mathcal{C}[\![\forall \underline{X}.\, \mathsf{B}]\!]_\gamma &= \{\kappa \in \prod_{\underline{A} \in \mathcal{A}} \mathcal{C}[\![\mathsf{B}]\!]_{\gamma[\underline{A}/\underline{X}]} \mid \forall \underline{A}, \underline{B} \in \mathcal{A},\, \forall Q \in \mathcal{R}_{\mathcal{A}}(\underline{A},\underline{B}).\ \mathcal{R}[\![\mathsf{B}]\!]_{\Delta_{\gamma}[Q/\underline{X}]}(\kappa_{\underline{A}}, \kappa_{\underline{B}})\}\ .
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{A}[\![\mathsf{B} \to \underline{\mathsf{A}}]\!]_\gamma &= \mathcal{A}[\![\underline{\mathsf{A}}]\!]_\gamma{}^{\mathcal{C}[\![\mathsf{B}]\!]_\gamma}\\
\mathcal{A}[\![\forall X.\, \underline{\mathsf{A}}]\!]_\gamma &= \{\pi \in \prod_{A \in \mathcal{C}} \mathcal{A}[\![\underline{\mathsf{A}}]\!]_{\gamma[A/X]} \mid \forall A, B \in \mathcal{C},\, \forall R \in \mathcal{R}_{\mathcal{C}}(A,B).\ \mathcal{R}[\![\underline{\mathsf{A}}]\!]_{\Delta_{\gamma}[R/X]}(\pi_A, \pi_B)\}\\
\mathcal{A}[\![\underline{X}]\!]_\gamma &= \gamma(\underline{X})\\
\mathcal{A}[\![\forall \underline{X}.\, \underline{\mathsf{A}}]\!]_\gamma &= \{\kappa \in \prod_{\underline{A} \in \mathcal{A}} \mathcal{A}[\![\underline{\mathsf{A}}]\!]_{\gamma[\underline{A}/\underline{X}]} \mid \forall \underline{A}, \underline{B} \in \mathcal{A},\, \forall Q \in \mathcal{R}_{\mathcal{A}}(\underline{A},\underline{B}).\ \mathcal{R}[\![\underline{\mathsf{A}}]\!]_{\Delta_{\gamma}[Q/\underline{X}]}(\kappa_{\underline{A}}, \kappa_{\underline{B}})\}\ .
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{R}[\![X]\!]_\rho(x_1,x_2) &\Leftrightarrow \rho_{\mathcal{R}}(X)(x_1,x_2)\\
\mathcal{R}[\![\mathsf{B} \to \mathsf{C}]\!]_\rho(f_1,f_2) &\Leftrightarrow \forall x_1 \in \mathcal{C}[\![\mathsf{B}]\!]_{\rho_1}, x_2 \in \mathcal{C}[\![\mathsf{B}]\!]_{\rho_2}.\ \mathcal{R}[\![\mathsf{B}]\!]_\rho(x_1,x_2) \implies \mathcal{R}[\![\mathsf{C}]\!]_\rho(f_1(x_1), f_2(x_2))\\
\mathcal{R}[\![\forall X.\, \mathsf{B}]\!]_\rho(\pi_1,\pi_2) &\Leftrightarrow \forall A_1, A_2 \in \mathcal{C}, \forall R \in \mathcal{R}_{\mathcal{C}}(A_1,A_2).\ \mathcal{R}[\![\mathsf{B}]\!]_{\rho[R/X]}((\pi_1)_{A_1}, (\pi_2)_{A_2})\\
\mathcal{R}[\![\underline{X}]\!]_\rho(x_1,x_2) &\Leftrightarrow \rho_{\mathcal{R}}(\underline{X})(x_1,x_2)\\
\mathcal{R}[\![\underline{\mathsf{A}} \multimap \underline{\mathsf{B}}]\!]_\rho(h_1,h_2) &\Leftrightarrow \forall x_1 \in \mathcal{C}[\![\underline{\mathsf{A}}]\!]_{\rho_1}, x_2 \in \mathcal{C}[\![\underline{\mathsf{A}}]\!]_{\rho_2}.\ \mathcal{R}[\![\underline{\mathsf{A}}]\!]_\rho(x_1,x_2) \implies \mathcal{R}[\![\underline{\mathsf{B}}]\!]_\rho(h_1(x_1), h_2(x_2))\\
\mathcal{R}[\![\forall \underline{X}.\, \mathsf{B}]\!]_\rho(\kappa_1,\kappa_2) &\Leftrightarrow \forall \underline{A}_1, \underline{A}_2 \in \mathcal{A}, \forall Q \in \mathcal{R}_{\mathcal{A}}(\underline{A}_1,\underline{A}_2).\ \mathcal{R}[\![\mathsf{B}]\!]_{\rho[Q/\underline{X}]}((\kappa_1)_{\underline{A}_1}, (\kappa_2)_{\underline{A}_2})\ .
\end{aligned}
$$

Figure 3. Interpretation of Types

$\mathcal{C}[\![\mathsf{B}]\!]_\gamma$ are the ones in $\mathcal{C}$, and those used in the definition of $\mathcal{A}[\![\underline{\mathsf{A}}]\!]_\gamma$ are those in $\mathcal{A}$, as (weakly) created by U. We write Δ_γ for the relational Θ-environment that maps X (resp. $\underline{X}$) to $\Delta_{\gamma(X)}$ (resp. $\Delta_{\gamma(\underline{X})}$). We also use an obvious notation for update of environments. The algebras defined by $\mathcal{A}[\![\forall Y.\, \underline{\mathsf{A}}]\!]_\gamma$ and $\mathcal{A}[\![\forall \underline{X}.\, \underline{\mathsf{A}}]\!]_\gamma$ are the canonical algebras carried by the subsets of the product algebras.

The caveat referred to above is that the interpretations of the polymorphic types in Figure 3 do not, on the face of it, make sense, since they involve products over proper classes of objects. Remarkably, the definitions are rescued by the fact that, in them, the fictitious products are cut down to their parametric elements. For example, using condition (C4), one can show that each tuple π satisfying the *parametricity property*:

$$\forall A, B \in \mathcal{C},\, \forall R \in \mathcal{R}_{\mathcal{C}}(A,B).\ \mathcal{R}[\![\mathsf{B}]\!]_{\Delta_{\gamma}[R/X]}(\pi_A, \pi_B)$$

is determined by its elements $\{\pi_A\}_{A \in \mathbf{C}}$. Similarly, by (A4), each parametric κ is determined by $\{\kappa_{\underline{A}}\}_{\underline{A} \in \mathbf{A}}$. Thus, in either case, there are only set-many parametric tuples. For space reasons, we omit the proof of this claim (which essentially goes back to [24]), preferring to focus on applications of the model rather than on technicalities in its construction.

Proposition 4.1 *$\mathcal{C}[\![\mathsf{B}]\!]_\gamma$, $\mathcal{A}[\![\underline{\mathsf{A}}]\!]_\gamma$ and $\mathcal{R}[\![\mathsf{B}]\!]_\rho$ are well defined by Figure 3. Further, for every computation type $\underline{\mathsf{A}}$, it holds that $\mathcal{C}[\![\underline{\mathsf{A}}]\!]_\gamma = U(\mathcal{A}[\![\underline{\mathsf{A}}]\!]_\gamma)$ and $\mathcal{R}[\![\underline{\mathsf{A}}]\!]_\rho \in \mathcal{R}_{\mathcal{A}}(\mathcal{A}[\![\underline{\mathsf{A}}]\!]_{\rho_1}, \mathcal{A}[\![\underline{\mathsf{A}}]\!]_{\rho_2})$.*

Lemma 4.2 (Identity extension) *For any type $\mathsf{B}(\Theta)$ and Θ-environment γ, it holds that $\mathcal{R}[\![\mathsf{B}]\!]_{\Delta_\gamma} = \Delta_{\mathcal{C}[\![\mathsf{B}]\!]_\gamma}$.*

Next, we define the interpretation of terms. Given a context Γ with all free type variables in Θ, a Θ-Γ-environment is a function defined on both the type variables in Θ and the term variables in Γ, such that the restriction of γ to Θ is a Θ-environment, and, for every type assigment $x\colon \mathsf{B}$ in Γ, it holds that $\gamma(x) \in \mathcal{C}[\![\mathsf{B}]\!]_\gamma$. A term $\Gamma \mid \Delta \vdash_\Theta t\colon \mathsf{B}$ (i.e. such that $\mathrm{ftv}(\Gamma, \Delta, t, \mathsf{B}) \subseteq \Theta$) is interpreted as an element $[\![t]\!]_\gamma \in \mathcal{C}[\![\mathsf{B}]\!]_\gamma$, relative to any Θ-(Γ,Δ)-environment γ. The definition of $[\![t]\!]_\gamma$ is given in Figure 4. In the two clauses that apply to $t(\underline{\mathsf{A}})$, we distinguish between the cases for t of type $\forall X. \mathsf{B}$ and $\forall \underline{X}. \mathsf{B}$. Note that the definition of $[\![s(t)]\!]_\gamma$ applies uniformly, whether s has type $\mathsf{B} \to \mathsf{C}$ or $\underline{\mathsf{A}} \multimap \underline{\mathsf{B}}$.

Proposition 4.3 *If $\Gamma \mid \Delta \vdash_\Theta t\colon \mathsf{B}$ then:*

1. *(Well-definedness) For any Θ-(Γ,Δ)-environment γ, the value $[\![t]\!]_\gamma \in \mathcal{C}[\![\mathsf{B}]\!]_\gamma$ is well defined.*

$$\begin{aligned}
[\![x]\!]_\gamma &= \gamma(x)\\
[\![\lambda x\colon \mathsf{B}.\, t]\!]_\gamma = [\![\lambda^\circ x\colon \underline{\mathsf{A}}.\, t]\!]_\gamma &= (d\colon \mathcal{C}[\![\mathsf{B}]\!]_\gamma \mapsto [\![t]\!]_{\gamma[d/x]})\\
[\![s(t)]\!]_\gamma &= [\![s]\!]_\gamma([\![t]\!]_\gamma)\\
[\![\Lambda X.\, t]\!]_\gamma &= \{[\![t]\!]_{\gamma[A/X]}\}_{A\in\mathcal{C}}\\
\llbracket t[\colon \forall X.\, \mathsf{B}](\mathsf{A})\rrbracket_\gamma &= ([\![t]\!]_\gamma)(\mathcal{C}[\![\mathsf{A}]\!]_\gamma)\\
[\![\Lambda \underline{X}.\, t]\!]_\gamma &= \{[\![t]\!]_{\gamma[\underline{A}/\underline{X}]}\}_{\underline{A}\in\mathcal{A}}\\
\llbracket t[\colon \forall \underline{X}.\, \mathsf{B}](\underline{\mathsf{A}})\rrbracket_\gamma &= ([\![t]\!]_\gamma)(\mathcal{A}[\![\underline{\mathsf{A}}]\!]_\gamma)
\end{aligned}$$

Figure 4. Interpretation of Terms

2. *(Relational invariance) For any relational* Θ-*environment* ρ, *and* Θ-(Γ,Δ)-*environments* γ_1,γ_2 *extending* ρ_1,ρ_2 *respectively, define*

   $$\mathcal{R}[\![\Gamma]\!]_\rho(\gamma_1,\gamma_2) \Leftrightarrow \forall x\colon \mathsf{A}\in(\Gamma,\Delta).\, \mathcal{R}[\![\mathsf{A}]\!]_\rho(\gamma_1(x),\gamma_2(x)).$$

   Then $\mathcal{R}[\![\Gamma]\!]_\rho(\gamma_1,\gamma_2)$ *implies* $\mathcal{R}[\![\mathsf{B}]\!]_\rho([\![t]\!]_{\gamma_1},[\![t]\!]_{\gamma_2})$.

If $\Gamma \mid x\colon \underline{\mathsf{A}} \vdash_\Theta t\colon \underline{\mathsf{B}}$ *then:*

3. *(Homomorphism property) For any* Θ-Γ-*environment* γ, *the function* $d\in\mathcal{C}[\![\underline{\mathsf{A}}]\!]_\gamma \mapsto [\![t]\!]_{\gamma[d/x]}$ *is a homomorphism from* $\mathcal{A}[\![\underline{\mathsf{A}}]\!]_\gamma$ *to* $\mathcal{A}[\![\underline{\mathsf{B}}]\!]_\gamma$.

Our main application of the model will be to establish equalities between terms. Henceforth, for $\Gamma \mid \Delta \vdash s\colon \mathsf{B}$ and $\Gamma \mid \Delta \vdash t\colon \mathsf{B}$, we write $\Gamma \mid \Delta \vdash s = t\colon \mathsf{B}$ to mean that $[\![s]\!]_\gamma = [\![t]\!]_\gamma$ for all appropriate γ.

5. Monadic types

In this section, we study the encoding of monadic types $!\,\mathsf{B}$ in our calculus, as defined by equation (1) of Section 1. One sees immediately that $!\,\mathsf{B}$ is always a computation type. We show that it enjoys the following derived introduction and elimination rules.

$$\frac{\Gamma \mid - \vdash t\colon \mathsf{B}}{\Gamma \mid - \vdash\, !\,t\colon !\,\mathsf{B}} \qquad \frac{\Gamma \mid \Delta \vdash t\colon !\,\mathsf{B} \qquad \Gamma, x\colon \mathsf{B} \mid - \vdash u\colon \underline{\mathsf{A}}}{\Gamma \mid \Delta \vdash \mathrm{let}\, !\,x\, \mathrm{be}\, t\, \mathrm{in}\, u\colon \underline{\mathsf{A}}}$$

Indeed, for this simply define:

$$\begin{aligned}
!\,t &=_{\mathrm{def}} \Lambda \underline{X}.\, \lambda p\colon \mathsf{B}\to\underline{X}.\, p(t)\\
\mathrm{let}\, !\,x\, \mathrm{be}\, t\, \mathrm{in}\, u &=_{\mathrm{def}} t(\underline{\mathsf{A}})(\lambda x\colon \mathsf{B}.\, u)\ .
\end{aligned}$$

It is the above rules that motivate our notation for the ! type constructor, since these are simply restrictions of the usual rules for the exponential ! of intuitionistic linear logic.

As a first application of relational parametricity for our system, we show that $!\,\mathsf{B}$ has the correct universal property for Moggi's monadic type. To keep the notation bearable, we frequently omit semantic brackets, treating syntactic objects as the semantic elements they define, and we freely mix syntactic expressions with semantic values. For example, given any set A in $\mathcal{C}$, we simply write $!A$ rather than $\mathcal{C}[\![!\,X]\!]_{[A/X]}$ or $\mathcal{A}[\![!\,X]\!]_{[A/X]}$, referring to $!A$ as a set or as an algebra respectively when disambiguation is needed.

Lemma 5.1 *1. If* $\Gamma \mid - \vdash t\colon \mathsf{B}$ *and* $\Gamma, x\colon \mathsf{B} \mid - \vdash u\colon \underline{\mathsf{A}}$ *then* $\Gamma \mid - \vdash \mathrm{let}\, !\,x\, \mathrm{be}\, !\,t\, \mathrm{in}\, u = u[t/x]\colon \underline{\mathsf{A}}$.

2. $\Gamma \mid y\colon !\,\mathsf{A} \vdash y = \mathrm{let}\, !x\, \mathrm{be}\, y\, \mathrm{in}\, !x\colon !\,\mathsf{A}$.

3. *Suppose that* $\Gamma \mid \Delta \vdash s\colon !\,\mathsf{A}$, $\Gamma, x\colon \mathsf{A} \mid - \vdash t\colon \underline{\mathsf{B}}$ *and* $\Gamma \mid y\colon \underline{\mathsf{B}} \vdash u\colon \underline{\mathsf{C}}$, *then* $\Gamma \mid \Delta \vdash \mathrm{let}\, !x\, \mathrm{be}\, s\, \mathrm{in}\, u[t/y] = u[\,\mathrm{let}\, !x\, \mathrm{be}\, s\, \mathrm{in}\, t\,/\,y]\colon \underline{\mathsf{C}}$.

Proof. Item 1 is a straightforward consequence of the semantic validity of beta equality.

For 2, we must show that $y = y(!\,\mathsf{A})(\lambda x\colon \mathsf{A}.\, !\,x)$ at type $\forall \underline{X}.\, (\mathsf{A}\to\underline{X})\to\underline{X}$. By evident extensionality properties of the model, it suffices to show that, for any algebra $\underline{B}$ and $f\colon \mathsf{A}\to U\underline{B}$ in $\mathcal{C}$, we have $y(\underline{B})(f) = y(!\,\mathsf{A})(\lambda x\colon \mathsf{A}.\, !\,x)(\underline{B})(f)$.

Consider the homomorphism $g\colon !\,\mathsf{A} \multimap \underline{B}$ defined by $g(z) = z(\underline{B})(f)$. Then $\langle g\rangle \in \mathcal{R}_{\mathcal{A}}(!\,\mathsf{A},\underline{B})$. So, by parametricity,

$$((\Delta_{\mathsf{A}}\to\langle g\rangle)\to\langle g\rangle)\,(y(!\,\mathsf{A}),\, y(\underline{B}))\ . \tag{5}$$

For any $x\in\mathsf{A}$, we have $g(!x) = (\Lambda\underline{X}.\, \lambda p.\, p(x))(\underline{B})(f) = f(x)$, i.e.

$$(\Delta_{\mathsf{A}}\to\langle g\rangle)\,(\lambda x\colon \mathsf{A}.\, !x,\, f)\ . \tag{6}$$

Combining (5) and (6), we obtain that

$$\langle g\rangle\,(y(!\,\mathsf{A})(\lambda x\colon \mathsf{A}.\, !x),\, y(\underline{B})(f))\ ,$$

i.e. $g(y(!\,\mathsf{A})(\lambda x\colon \mathsf{A}.\, !x)) = y(\underline{B})(f)$. Thus it indeed holds that $y(!\,\mathsf{A})(\lambda x\colon \mathsf{A}.\, !\,x)(\underline{B})(f) = y(\underline{B})(f)$.

For 3, $h = \lambda^\circ y\colon \underline{\mathsf{B}}.\, u\colon \underline{\mathsf{B}} \multimap \underline{\mathsf{C}}$ is a homomorphism, so $\langle h\rangle \in \mathcal{R}_{\mathcal{A}}(\underline{\mathsf{B}},\underline{\mathsf{C}})$. By parametricity, we have that

$$((\Delta_{\mathsf{A}}\to\langle h\rangle)\to\langle h\rangle)\,(s(\underline{\mathsf{B}}),\, s(\underline{\mathsf{C}}))\ . \tag{7}$$

Consider $\lambda x\colon \mathsf{A}.\, t\colon \mathsf{A}\to\underline{\mathsf{B}}$ and $\lambda x\colon \mathsf{A}.\, u[t/y]\colon \mathsf{A}\to\underline{\mathsf{C}}$. Then, for $x\in\mathsf{A}$, it holds that $h((\lambda x\colon \mathsf{A}.\, t)(x)) = u[t/y] = (\lambda x\colon \mathsf{A}.\, u[t/y])(x)$, i.e.

$$(\Delta_{\mathsf{A}}\to\langle h\rangle)\,(\lambda x\colon \mathsf{A}.\, t,\, \lambda x\colon \mathsf{A}.\, u[t/y])\ . \tag{8}$$

Combining (7) and (8), we obtain that

$$\langle h\rangle\,(s(\underline{\mathsf{B}})(\lambda x\colon \mathsf{A}.\, t),\, s(\underline{\mathsf{C}})(\lambda x\colon \mathsf{A}.\, u[t/y]))\ ,$$

i.e. $h(s(\underline{\mathsf{B}})(\lambda x\colon \mathsf{A}.\, t)) = s(\underline{\mathsf{C}})(\lambda x\colon \mathsf{A}.\, u[t/y])$. So indeed we have $u[\,\mathrm{let}\, !x\, \mathrm{be}\, s\, \mathrm{in}\, t\,/\,y] = h(s(\underline{\mathsf{B}})(\lambda x\colon \mathsf{A}.\, t)) = s(\underline{\mathsf{C}})(\lambda x\colon \mathsf{A}.\, u[t/y]) = \mathrm{let}\, !x\, \mathrm{be}\, s\, \mathrm{in}\, u[t/y]$. $\square$

For any set A in $\mathcal{C}$ define $\eta_A\colon A\to !A$ by $\eta_A = \lambda x.\, !x$.

Theorem 5.2 *The function $\eta_A \colon A \to !A$ presents $!A$ as the free algebra over A, i.e. for any algebra $\underline{A}$ and function $f \colon A \to U\underline{A}$, there exists a unique homomorphism $h \colon !A \multimap \underline{A}$ such that $h \circ \eta_A = f$. Indeed, h is given by $\lambda^\circ y.\ \mathrm{let}\ !x\ \mathrm{be}\ y\ \mathrm{in}\ f(x)$.*

Proof. Clearly $\lambda^\circ y.\ \mathrm{let}\ !x\ \mathrm{be}\ y\ \mathrm{in}\ f(x)$ is a homomorphism, and $(\lambda^\circ y.\ \mathrm{let}\ !x\ \mathrm{be}\ y\ \mathrm{in}\ f(x)) \circ \eta_A = f$ because $\mathrm{let}\ !x\ \mathrm{be}\ !x\ \mathrm{in}\ f(x) = f(x)$ by Lemma 5.1.1. For uniqueness, suppose h is such that $h \circ \eta_A = f$. Then

$$\begin{aligned} h(y) &= h(\mathrm{let}\ !\,x\ \mathrm{be}\ y\ \mathrm{in}\ !x) && \text{(Lemma 5.1.2)} \\ &= \mathrm{let}\ !\,x\ \mathrm{be}\ y\ \mathrm{in}\ h(!x) && \text{(Lemma 5.1.3)} \\ &= \mathrm{let}\ !\,x\ \mathrm{be}\ y\ \mathrm{in}\ f(x) && (h \circ \eta_A = f)\ , \end{aligned}$$

as required. □

It follows from the above theorem that the operation mapping A to the algebra $!A$ is the object part of a functor $F \colon \mathcal{C} \to \mathcal{A}$ left adjoint to U. We write T for the associated monad UF on $\mathcal{C}$.

The bijective correspondence of Theorem 5.2 can be expressed in the type theory PE as an isomorphism of (value) types between $!\mathsf{A} \multimap \underline{\mathsf{B}}$ and $\mathsf{A} \to \underline{\mathsf{B}}$. Thus we have a Girard decomposition of function spaces with computation type codomains, further motivating the ! notation.

We end this section with a characterisation of the induced relational lifting of the ! type constructor.

Proposition 5.3 *Suppose A, B are objects of $\mathcal{C}$ and $R \colon \mathcal{R}_{\mathcal{C}}(A, B)$ is a relation. Then $!\,R \colon \mathcal{R}_{\mathcal{A}}(!\,A, !\,B)$ is the smallest admissible $\mathcal{A}$-relation containing all pairs of the form $(\eta(x), \eta(y))$ for $(x, y) \in R$.*

6. Specialising the calculus to specific effects

The type theory PE is a generic calculus for effects since the type $!\,\mathsf{B}$ can be interpreted as an arbitrary monad, and no further effect-specific features are included. In this regard, PE is analogous to Moggi's computational λ-calculus [12], computational metalanguage [13] and Levy's call-by-push-value [9]. As with those calculi, specific effects can be incorporated by specialising the calculus appropriately. In this section we consider various such specialisations, emphasising, in particular, the interaction with parametricity.

In a recent programme of research [20], Plotkin and Power have shown that many monads of computational interest can be profitably viewed as free algebra constructions for equational theories. This approach arises naturally from a computational viewpoint: the "algebraic operations" used to specify the theory correspond to programming primitives that cause effects, and the equational theory simply expresses natural behavioural equivalences between such primitives. We begin this section with an analysis of how to specialise PE to the case of such "algebraic effects".

Our approach is justified by a general theorem, which we now present. As one of their central results about algebraic effects, Plotkin and Power establish a one-to-one correspondence between "algebraic operations" and (what they call) "generic effects" [19]. The theorem below reformulates this correspondence in our setting, and adds a third equivalent induced by our polymorphic description of monadic types. We shall apply this third equivalent to obtain the correct polymorphic typing for algebraic operations in effect-specific specialisations of PE.

Theorem 6.1 *For any set A in $\mathcal{C}$, there are one-to-one correspondences between:*

1. *"algebraic operations of arity A", i.e. natural transformations from the functor $(U(-))^A \colon \mathcal{A} \to \mathcal{C}$ to U,*
2. *"generic effects over A", i.e. elements of TA, and*
3. *"polymorphic computation type operations of arity A", i.e. elements of $\forall \underline{X}.\ (A \to \underline{X}) \to \underline{X}$.*

The simplifications in the formulation of statement 1 above, compared with [19], are due to our set-theoretic setting, which renders it unnecessary to consider issues relating to enrichment or tensorial strength. Also note that, by statement 2, the other two statements, in spite of appearances, depend only on the monad T on $\mathcal{C}$, not on how it is resolved into an adjunction $F \dashv U \colon \mathcal{A} \to \mathcal{C}$.

To illustrate how Theorem 6.1 informs the specialisation of PE to algebraic effects, we consider nondeterminism as a typical example. As in [20], nondeterministic choice is naturally formulated using a binary operation "or" satisfying the semilattice equations:

$$x \,\mathrm{or}\, x = x, \quad x \,\mathrm{or}\, y = y \,\mathrm{or}\, x, \quad x \,\mathrm{or}\, (y \,\mathrm{or}\, z) = (x \,\mathrm{or}\, y) \,\mathrm{or}\, z\ .$$

Define the category $\mathcal{A}_{\mathrm{nd}}$ of "nondeterministic algebras" to have, as objects, structures (A, or_A) where A is a set in $\mathcal{C}$ and $\mathrm{or}_A \colon A \times A \to A$ satisfies the semilattice equations, and, as morphisms from (A, or_A) to (B, or_B), functions from A to B that are homomorphisms with respect to the "or" operations. It is easily verified that the obvious forgetful functor $U \colon \mathcal{A}_{\mathrm{nd}} \to \mathcal{C}$ satisfies conditions (A1)–(A4).

Since the morphisms in $\mathcal{A}_{\mathrm{nd}}$ are homomorphisms, the operation mapping any nondeterministic algebra (A, or_A) to the function $\mathrm{or}_A \colon A^2 \to A$ is an algebraic operation of arity 2 in the sense of statement 1 of Theorem 6.1. Thus, applying Theorem 6.1 and currying, one obtains a corresponding polymorphic operation:

$$\mathrm{or} : \ \forall \underline{X}.\ \underline{X} \to \underline{X} \to \underline{X}\ .$$

Accordingly, nondeterministic choice can be incorporated in PE by adding a constant or, typed as above, to the type theory. This example illustrates the general pattern for adding algebraic operations as polymorphic constants to our type theory, and readily adapts to the algebraic operations associated with other algebraic effects.

A limitation of the notion of algebraic operation is that there exist effect-specific programming primitives that are not algebraic operations. One well-known example of such a primitive is exception handling. Below, we show how exception handling may also be incorporated within our approach as a suitably typed polymorphic constant. The approach is justified by a general theorem, giving another instance of a coincidence between natural transformations and elements of polymorphic type.

Theorem 6.2 *For any $n \in \mathbb{N}$, there are one-to-one correspondences between:*

1. *Natural transformations from $(F(-))^n : \mathcal{C} \to \mathcal{A}$ to $F : \mathcal{C} \to \mathcal{A}$, and*
2. *elements of $\forall X.\, (n \to !X) \multimap !X$,*

where, in statement 2, we write n for the n-fold coproduct type $1 + \cdots + 1$, as defined in Figure 2.

We now consider exception handling in detail. We assume we have a set E of exceptions with decidable equality (i.e. for all $e, e' \in E$ either $e = e'$ or $e \neq e'$). We also assume (for simplicity) that $\mathcal{C}$ is closed under binary coproduct in $\mathbf{Set}$. We define the category $\mathcal{A}_{\text{exc}}$ of "exception algebras" to have, as objects, structures $(A, \{\mathrm{raise}^e_A\}_{e \in E})$ where $\mathrm{raise}^e_A \in A$, and, as morphisms from $(A, \{\mathrm{raise}^e_A\}_{e \in E})$ to $(B, \{\mathrm{raise}^e_B\}_{e \in E})$, functions from A to B that map each raise^e_A to raise^e_B. Since the raise_e elements are algebraic constants (operations of arity 0), they can be added to PE as constants:

$$\mathrm{raise}^e : \forall \underline{X}.\, \underline{X} \ .$$

As is standard, the forgetful functor functor from $\mathcal{A}_{\text{exc}}$ to $\mathcal{C}$, has as its left adjoint the functor F mapping A to the exception algebra $(A + E, \{\mathrm{inr}(e)\}_{e \in E})$. For an exception $e \in E$, the handling operation over A is the function $\mathrm{handle}^e_A : (F(A))^2 \to F(A)$ defined by

$$\mathrm{handle}^e_A(p, q) \;=\; \begin{cases} p & \text{if } p \neq \mathrm{inr}(e) \\ q & \text{if } p = \mathrm{inr}(e) \ . \end{cases}$$

It is easily shown that this specifies a natural transformation from $(F(-))^2 : \mathcal{C} \to \mathcal{A}_{\text{exc}}$ to $F : \mathcal{C} \to \mathcal{A}_{\text{exc}}$. In particluar, the component handle^e_A of the natural transformation does lie in $\mathcal{A}_{\text{exc}}$ because the interpretation of raise^e in the exception algebra $F(A)^2$ is the pair $(\mathrm{inr}(e), \mathrm{inr}(e))$. Thus, by Theorem 6.2, exception handling can be incorporated in PE by adding typed constants:

$$\mathrm{handle}^e : \forall X.\, (2 \to !X) \multimap !X \ .$$

The main surprise with this typing is that exception handling is given a "linear" type. From this typing, one of course obtains an associated term of the expected (but less informative) type $\forall X.\, (2 \to !X) \to !X$.

Both Theorems 6.1 and 6.2 relate elements of certain polymorphic types with natural transformations between associated functors. In fact, more generally, for types that determine functors, parametricity implies naturality (cf. [18]). However, the exact correspondences between natural transformations and parametric elements established above depend heavily on the precise forms of types considered there.

The forms of n-ary operation considered in this section by no means exhaust the collection of operations of interest from an effects perspective. Control operators provide a particularly interesting class of examples, since their associated continuations monads do not naturally fit into the Plotkin-Power framework for algebraic effects. One way of specialising PE to the case of control is discussed briefly in the next section.

7. Relation to other systems

Several computational effects of interest, including nontermination, nondeterminism, and probabilistic choice, give rise to monads on $\mathcal{C}$ that are *commutative*, cf. [13]. The collection of models of PE in which $\mathcal{A}$ is the category of algebras for a commutative monad T is of special interest since, for such monads, the set of homomorphisms $\underline{A} \multimap \underline{B}$ between algebras $\underline{A}, \underline{B}$ carries a natural algebra structure which provides a closed structure on the category $\mathcal{A}$. For such models, it is thus natural to modify our type system by including $\underline{\mathsf{A}} \multimap \underline{\mathsf{B}}$ as a computation type. Making this adjustment, one obtains second-order intuitionistic linear type theory as the fragment of computation types:

$$\underline{X} \mid \underline{\mathsf{A}} \multimap \underline{\mathsf{B}} \mid \underline{\mathsf{A}} \to \underline{\mathsf{B}} \mid \forall \underline{X}.\, \underline{\mathsf{A}} \ . \tag{9}$$

Thus we obtain a rich collection of models for the type theory proposed by Plotkin as a foundation for combining polymorphism and recursion [17].

By the above, PE is naturally viewed as a generalisation of second-order intuitionistic linear type theory valid in a wider collection of models. A remarkable feature of second-order intuitionistic linear type theory, due to Plotkin, is that a rich collection of type constructors can be defined in terms of the three primitives in (9) above, cf. [17, 1, 2]. In fact, using well chosen variants of Plotkin's definitions, a similar richness of definability is available in PE, see Figure 5 (which makes use of the definitions in Figure 2). We briefly discuss these encodings.

Semantically, because $U : \mathcal{A} \to \mathcal{C}$ weakly creates limits, algebras are closed under products in $\mathcal{C}$. Syntactically, however, the types 1 and $\underline{\mathsf{A}} \times \underline{\mathsf{B}}$ from Figure 2 are *not* computation types. Thus the alternative encodings 1° and $\underline{\mathsf{A}} \times^\circ \underline{\mathsf{B}}$ are needed to obtain products of computation types as computation types. The types 0° and $\underline{\mathsf{A}} \oplus \underline{\mathsf{B}}$ from Figure 5 define respectively an initial object and binary coproduct in the category $\mathcal{A}$. This structure in *not* preserved by U, and coproducts of algebras behave very differently from coproducts of sets in $\mathcal{C}$. (The latter are implemented by the sum types in Figure 2.) The type $\mathsf{B} \cdot \underline{\mathsf{A}}$ defines a $\mathcal{C}[\![\mathsf{B}]\!]$-fold copower of $\mathcal{A}[\![\underline{\mathsf{A}}]\!]$ in $\mathcal{A}$. Figure 5 also contains: existential types, $\exists^\circ X.\, \underline{\mathsf{A}}$ and $\exists^\circ \underline{X}.\, \underline{\mathsf{A}}$, packaged up as computation types; inductive computation types, $\mu^\circ \underline{X}.\, \underline{\mathsf{A}}$; and coinductive computation types, $\nu^\circ \underline{X}.\, \underline{\mathsf{A}}$. As is standard, the (co)inductive types rely on the functoriality of type expressions in their positive arguments. It is a consequence of relational parametricity that the above types all enjoy the correct universal properties. The arguments are carried out most naturally using a suitable logic for relational parametricity in PE, and will appear in a forthcoming paper [11].

A simple application of Figures 2 and 5 is to translate Levy's CBPV calculus [9] into PE. For this, coproducts and products of value types are translated using $+$ and $\times$ from Figure 2, products of computation types are translated using $\times^\circ$ from Figure 5, Levy's F constructor is translated using $!$, and U is simply ignored.

Finally, we mention the case of control, which was one of the motivations for this work. Control primitives can be modelled naturally within PE by adding a polymorphic constant of type (using 0° from Figure 5):

$$\forall \underline{X}.\, ((\underline{X} \multimap 0^\circ) \to 0^\circ) \;\multimap\; \underline{X} \;.$$

The resulting theory is studied in a companion article [10], where it is shown that Hasegawa's [5] results on polymorphic definability the second-order $\lambda\mu$-calculus fall out as special cases of constructions from Figure 5.

More generally, the theory developed in this paper should be applicable whenever there is interaction between polymorphism and effects. This is a topic for future work.

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

Figure 5. Definable computation types

References

[1] G. Bierman, A. Pitts, and C. Russo. Operational properties of Lily, a polymorphic linear lambda calculus with recursion. *ENTCS*, 41:70–88, 2000.

[2] L. Birkedal, R. E. Møgelberg, and R. L. Petersen. Linear Abadi & Plotkin logic. *Log. Meth. in Comp. Sci.*, 2, 2006.

[3] J.-Y. Girard. On the unity of logic. *Annals of Pure and Applied Logic*, 59:201–217, 1993.

[4] J. Goubault-Larrecq, S. Lasota, and D. Nowak. Logical relations for monadic types. In *Computer Science Logic*, pages 553–568. Springer LNCS 2571, 2002.

[5] M. Hasegawa. Relational parametricity and control. *Logical Methods in Computer Science*, 2, 2006. Special issue for selected papers from LICS 2005.

[6] J.M.E. Hyland. A small complete category. *Annals of Pure and Applied Logic*, 40:135 – 165, 1988.

[7] J.M.E. Hyland, E. Robinson, and G. Rosolini. The discrete objects in the effective topos. *Proc. LMS.*, 3(60), 1990.

[8] S. Katsumata. A semantic formulation of $\top\top$-lifting and logical predicates for computational metalanguage. In *Computer Science Logic*, Springer LNCS 3634, 2005.

[9] P. Levy. *Call-By-Push-Value*. Springer, 2004.

[10] R.E. Møgelberg and A. Simpson. Relational Parametricity for Control Considered as a Computational Effect. Submitted 2006. At http://homepages.inf.ed.ac.uk/als/Research/

[11] R.E. Møgelberg and A. Simpson. A logic for parametric polymorphism with effects. In preparation, 2007.

[12] E. Moggi. Computational lambda-calculus and monads. In *LICS'89, Proc. 4th LICS Symposium*, pages 14–23, 1989.

[13] E. Moggi. Notions of computation and monads. *Information and Computation*, 93(1), 1991.

[14] M. Parigot. Strong normalization for second order classical natural deduction. *J. Symb. Logic*, 62:1461–1479, 1997.

[15] A. Pitts. Polymorphism is set theoretic, constructively. In *Proc. CTCS*, pages 12–39. Springer LNCS 283, 1987.

[16] A. Pitts. Non-trivial power types can't be subtypes of polymorphic types. In *Proc. 4th LICS Symp.*, pages 6–13, 1989.

[17] G. Plotkin. Type theory and recursion (extended abstract). In *Proc. 8th LICS Symposium*, page 374, 1993.

[18] G. Plotkin and M. Abadi. A logic for parametric polymorphism. *Proc. TLCA*, pp.361–375. Springer LNCS 664, 1993.

[19] G. Plotkin and A.J. Power. Algebraic operations and generic effects. *Applied categorical Structures*, 11:69–94, 2003.

[20] G. Plotkin and A.J. Power. Computational effects and operations: an overview. *ENTCS*, 73:149–163, 2004.

[21] J. Reynolds. Types, abstraction and parametric polymorphism. In *Inf. Processing*, pp.513–523. N. Holland, 1983.

[22] J. Reynolds. Polymorphism is not set-theoretic. In *Semantics of Data Types*. Springer LNCS 173, 1984.

[23] E. Robinson. How complete is PER? In *Proc. 4th LICS Symposium*, pages 106–111, 1989.

[24] G. Rosolini and A. Simpson. *Using Synthetic Domain Theory to Prove Operational Properties of a Polymorphic Programming Language Based on Strictness*. Preprint, 2004.

[25] A. Ščedrov. Intuitionistic set theory. In *Harvey Friedman's Research on The Foundations of Mathematics*, pages 257–284. Elsevier Science Publishers, 1985.

[26] P. Wadler. Theorems for free! In *Proc. 4th Int. Conf. on Funct. Prog. Languages and Computer Arch.* London, 1989.

A. Selected proofs

For the benefit of the referees, this appendix contains a few ommitted lemmas, and outlines arguments for the main missing proofs.

A.1 Section 2

The following simple lemmas state basic properties of the type system.

Lemma A.1 (Unicity of types) *For any* Γ, Δ, t *there is at most one type* B *such that* $\Gamma \mid \Delta \vdash t \colon \mathsf{B}$.

Lemma A.2 (Substitution)

1. *If* $\Gamma, x \colon \mathsf{A} \mid \Delta \vdash t \colon \mathsf{B}$ *and* $\Gamma \mid - \vdash s \colon \mathsf{A}$ *then* $\Gamma \mid \Delta \vdash t[s/x] \colon \mathsf{B}$.
2. *If* $\Gamma \mid x \colon \underline{\mathsf{A}} \vdash t \colon \underline{\mathsf{B}}$ *and* $\Gamma \mid \Delta \vdash s \colon \underline{\mathsf{A}}$ *then* $\Gamma \mid \Delta \vdash t[s/x] \colon \underline{\mathsf{B}}$.

A.2 Section 3

Axiom (A1) gives a way of picking representatives in $\mathcal{A}$ for subalgebras presented by subsets:

Lemma A.3 *For each* $A \in \mathrm{Sub}_{\mathcal{A}}(\underline{A})$ *there is a specified algebra* $\underline{B}$ *and mono* $f \colon \underline{B} \rightarrowtail \underline{A}$ *in* $\mathcal{A}$ *such that* Uf *is the inclusion of* A *into* $U\underline{A}$.

Proof. Suppose $A \subseteq U\underline{A}$ carries a subalgebra of $\underline{A}$. Then the set

$$\{(\underline{B}, i) \mid \underline{B} \in \mathbf{A}, i \colon \underline{B} \multimap \underline{A} \text{ mono}, U(i) \cong (A \subseteq U\underline{A})\} \tag{10}$$

where the last isomorphism is an isomorphism of subobjects, is non-empty. The set (10) is a diagram in $\mathcal{A}$, and A is a limit in $\mathcal{C}$ of U applied to this diagram. Now, (A1) gives the specified mono projecting to $A \subseteq U\underline{A}$. □

Lemma A.4 *If* $\mathcal{C}$ *satisfies (C1)–(C4) and* $U \colon \mathcal{A} \to \mathcal{C}$ *satisfies (A1)–(A4) then the collections* $\mathcal{R}_{\mathcal{C}}(A, B) = \mathrm{Sub}_{\mathcal{C}}(A \times B)$ *and* $\mathcal{R}_{\mathcal{A}}(\underline{A}, \underline{B}) = \mathrm{Sub}_{\mathcal{A}}(\underline{A} \times \underline{B})$ *satisfy (R1)–(R4).*

Proof. We just show that $\mathrm{Sub}_{\mathcal{A}}(\underline{A}, \underline{B})$ is closed under intersections. So suppose we are given a set $(Q_i)_{i\in I}$ of subsets in $\mathrm{Sub}_{\mathcal{A}}(\underline{A}, \underline{B})$. We need to show that the subset $\bigcap_i Q_i \subseteq U\underline{A} \times U\underline{B}$ carries a subalgebra of $\underline{A} \times \underline{B}$. Denote for each $i \in I$ by $q_i \colon Q_i' \multimap \underline{A} \times \underline{B}$ the mono in $\mathcal{A}$ above the inclusion $Q_i \subseteq U\underline{A} \times U\underline{B}$ as specified by Lemma A.3. Then the limit of the diagram given by the q_i as weakly created by U is a subalgebra of $\underline{A} \times \underline{B}$ above $\bigcap_i Q_i \subseteq U\underline{A} \times U\underline{B}$. □

Proof of Proposition 3.1. Arguments that (A1)–(A4) are satisfied exist in the main text, and (R1)–(R4) are satisfied by Lemma A.4. □

A.3 Section 4

In Section 4, we have formulated the interpretation of polymorphic types using products over the collections of objects in $\mathcal{C}$ and $\mathcal{A}$. Strictly speaking, such products do not exist in the set theory, because the collections in question are not sets but proper classes. We have chosen to give this presentation in the main text for reasons of space, but here we show how to make sense of the large products in the set theoretic framework.

The easiest way to do this is to redefine the interpretations of polymorphic types by taking, as the "official" definition, products over the sets $\mathbf{C}$ and $\mathbf{A}$ rather than over the classes $\mathcal{C}$ and $\mathcal{A}$. The relevant modifications to Figure 3 are detailed in Figure 6.

Below, we shall explain how the new definition can indeed be equivalently viewed as being given in terms of the large product of Figure 3. First, however, we prove Proposition 4.1 for the official definition, i.e. that the definition is well defined.

Proof of Proposition 4.1 (sketch). The proof of well definedness is by induction over the structure of types. We focus first on showing that the relational interpretation of types defines admissible relations. Notice first that the relation $\mathcal{R}[\![\mathsf{B} \to \mathsf{C}]\!]_\rho$ can be rewritten as

$$\bigcap_{(x_1,x_2)\in\mathcal{R}[\![\mathsf{B}]\!]_\rho} (\mathrm{ev}_{x_1}, \mathrm{ev}_{x_2})^{-1}\mathcal{R}[\![\mathsf{C}]\!]_\rho$$

where ev_{x_1} denotes the map from $\mathcal{R}[\![\mathsf{B} \to \mathsf{C}]\!]_{\rho_1}$ to $\mathcal{R}[\![\mathsf{C}]\!]_{\rho_1}$ given by evaluation at x_1, and ev_{x_2} is defined likewise. For value types B, C it follows that $\mathcal{R}[\![\mathsf{B} \to \mathsf{C}]\!]_\rho$ is an admissible $\mathcal{C}$ relation from the induction hypothesis and (R2) and (R3). If C is a computation type, $\mathsf{B} \to \mathsf{C}$ becomes a computation type and we must check that $\mathcal{R}[\![\mathsf{B} \to \mathsf{C}]\!]_\rho$ is an admissible $\mathcal{A}$ relation. Since the object $\mathcal{A}[\![\mathsf{B} \to \mathsf{C}]\!]_{\rho_1}$ is defined as a product $\mathcal{A}[\![\mathsf{B}]\!]_{\rho_1}^{\mathcal{C}[\![\mathsf{C}]\!]_{\rho_1}}$ in $\mathcal{A}$ and the evaluation map ev_{x_1} is the projection, it is a homomorphism. So again $\mathcal{R}[\![\mathsf{B} \to \mathsf{C}]\!]_\rho$ being admissible follows from the induction hypothesis and (R2), (R3). The proof of the other induction cases are similar.

To prove well definedness of $\mathcal{A}[\![\forall X.\ \underline{\mathsf{A}}]\!]_\gamma$ notice first that the formula in Figure 6 defines an element in $\mathrm{Sub}_{\mathcal{A}}(\prod_{A\in\mathbf{C}} \mathcal{A}[\![\underline{\mathsf{A}}]\!]_{\gamma[A/X]})$ since it can be exhibited as an intersection of $\mathcal{A}$-subobjects as above. We define $\mathcal{A}[\![\forall X.\ \underline{\mathsf{A}}]\!]_\gamma$ to be the specified $\mathcal{A}$ object representing the subset as given by Lemma A.3, thus defining $\mathcal{A}[\![\forall X.\ \underline{\mathsf{A}}]\!]_\gamma$ up to identity and not just up to isomorphism. □

$$\mathcal{C}[\![\forall X.\, \mathsf{B}]\!]_\gamma = \{\pi \in \prod_{A \in \mathbf{C}} \mathcal{C}[\![\mathsf{B}]\!]_{\gamma[A/X]} \mid \forall A, B \in \mathbf{C},\, \forall R \in \mathcal{R}_{\mathcal{C}}(A,B).\, \mathcal{R}[\![\mathsf{B}]\!]_{\Delta_\gamma[R/X]}(\pi_A, \pi_B)\}$$
$$\mathcal{C}[\![\forall \underline{X}.\, \mathsf{B}]\!]_\gamma = \{\kappa \in \prod_{\underline{A} \in \mathbf{A}} \mathcal{C}[\![\mathsf{B}]\!]_{\gamma[\underline{A}/\underline{X}]} \mid \forall \underline{A}, \underline{B} \in \mathbf{A},\, \forall Q \in \mathcal{R}_{\mathcal{A}}(\underline{A},\underline{B}).\, \mathcal{R}[\![\mathsf{B}]\!]_{\Delta_\gamma[Q/\underline{X}]}(\kappa_{\underline{A}}, \kappa_{\underline{B}})\}\ .$$

$$\mathcal{A}[\![\forall X.\, \underline{\mathsf{A}}]\!]_\gamma = \{\pi \in \prod_{A \in \mathbf{C}} \mathcal{A}[\![\underline{\mathsf{A}}]\!]_{\gamma[A/X]} \mid \forall A, B \in \mathbf{C},\, \forall R \in \mathcal{R}_{\mathcal{C}}(A,B).\, \mathcal{R}[\![\underline{\mathsf{A}}]\!]_{\Delta_\gamma[R/X]}(\pi_A, \pi_B)\}$$
$$\mathcal{A}[\![\forall \underline{X}.\, \underline{\mathsf{A}}]\!]_\gamma = \{\kappa \in \prod_{\underline{A} \in \mathbf{A}} \mathcal{A}[\![\underline{\mathsf{A}}]\!]_{\gamma[\underline{A}/\underline{X}]} \mid \forall \underline{A}, \underline{B} \in \mathbf{A},\, \forall Q \in \mathcal{R}_{\mathcal{A}}(\underline{A},\underline{B}).\, \mathcal{R}[\![\underline{\mathsf{A}}]\!]_{\Delta_\gamma[Q/\underline{X}]}(\kappa_{\underline{A}}, \kappa_{\underline{B}})\}\ .$$

$$\mathcal{R}[\![\forall X.\, \mathsf{B}]\!]_\rho(\pi_1, \pi_2) \Leftrightarrow \forall A_1, A_2 \in \mathbf{C}, \forall R \in \mathcal{R}_{\mathcal{C}}(A_1, A_2).\, \mathcal{R}[\![\mathsf{B}]\!]_{\rho[R/X]}((\pi_1)_{A_1}, (\pi_2)_{A_2})$$
$$\mathcal{R}[\![\forall \underline{X}.\, \mathsf{B}]\!]_\rho(\kappa_1, \kappa_2) \Leftrightarrow \forall \underline{A}_1, \underline{A}_2 \in \mathbf{A}, \forall Q \in \mathcal{R}_{\mathcal{A}}(\underline{A}_1, \underline{A}_2).\, \mathcal{R}[\![\mathsf{B}]\!]_{\rho[Q/\underline{X}]}((\kappa_1)_{\underline{A}_1}, (\kappa_2)_{\underline{A}_2})\ .$$

Figure 6. Redefinition of interpretations of polymorphic types

With the new definition, for all $\pi \in \mathcal{C}[\![\forall X.\, \mathsf{B}]\!]_\gamma$, where $\forall X.\, \mathsf{B}$ has type variables in Θ and γ is a Θ-environment, by definition, π contains a value π_B for each set $B \in \mathbf{C}$. However, to make sense of the large product used in Figure 3, we shall need to project π to an arbitrary set A in $\mathcal{C}$. Similarly, given $\kappa \in \mathcal{C}[\![\forall \underline{X}.\, \mathsf{B}]\!]_\gamma$, we have a value $\kappa_{\underline{B}}$ for each algebra $\underline{B} \in \mathbf{A}$, but we shall need to project κ to an arbitrary algebra $\underline{A}$ in $\mathcal{A}$. For the definitions of these projections we need the following lemma.

Lemma A.5 (Groupoid action) *For any* $\mathsf{B}(\Theta, X)$*, any* Θ*-environment* γ*, and isomorphism* $i\colon A \to B$ *in* $\mathcal{C}$*, there exists a unique isomorphism*

$$\mathrm{gpd}[\![\mathsf{B}]\!]_\gamma(i)\colon \mathcal{C}[\![\mathsf{B}]\!]_{\gamma[A/X]} \to \mathcal{C}[\![\mathsf{B}]\!]_{\gamma[B/X]}$$

such that

$$\mathcal{R}[\![\mathsf{B}]\!]_{\Delta_\gamma[\langle i\rangle/X]} \;=\; \langle \mathrm{gpd}[\![\mathsf{B}]\!]_\gamma(i)\rangle\ .$$

Moreover, if B *is a computation type then* $\mathrm{gpd}[\![\mathsf{B}]\!]_\gamma(i)$ *is a homomorphism from* $\mathcal{A}[\![\mathsf{B}]\!]_{\gamma[A/X]}$ *to* $\mathcal{A}[\![\mathsf{B}]\!]_{\gamma[B/X]}$.

Similarly, for any $\mathsf{B}(\Theta, \underline{X})$ *and isomorphism* $j\colon \underline{A} \multimap \underline{B}$*, there exists a unique isomorphism*

$$\mathrm{gpd}[\![\mathsf{B}]\!]_\gamma(j)\colon \mathcal{C}[\![\mathsf{B}]\!]_{\gamma[\underline{A}/\underline{X}]} \to \mathcal{C}[\![\mathsf{B}]\!]_{\gamma[\underline{B}/\underline{X}]}$$

such that

$$\mathcal{R}[\![\mathsf{B}]\!]_{\Delta_\gamma[\langle j\rangle/\underline{X}]} \;=\; \langle \mathrm{gpd}[\![\mathsf{B}]\!]_\gamma(j)\rangle\ .$$

Moreover, if B *is a computation type then* $\mathrm{gpd}[\![\mathsf{B}]\!]_\gamma(j)$ *is a homomorphism from* $\mathcal{A}[\![\mathsf{B}]\!]_{\gamma[\underline{A}/\underline{X}]}$ *to* $\mathcal{A}[\![\mathsf{B}]\!]_{\gamma[\underline{B}/\underline{X}]}$.

The mappings $i \mapsto \mathrm{gpd}[\![\mathsf{B}]\!]_\gamma(i)$ *and* $j \mapsto \mathrm{gpd}[\![\mathsf{B}]\!]_\gamma(j)$ *are functorial.*

Proof (sketch). The isomorphism $\mathrm{gpd}[\![\mathsf{B}]\!]_\gamma(i)$ is constructed by induction over the structure of B, by instantiating the functorial interpretation of B with i in positive occurences of X and i^{-1} in negative. For example $\mathrm{gpd}[\![\mathsf{B} \to \mathsf{C}]\!]_\gamma(i)$ maps $f \in \mathcal{C}[\![\mathsf{B} \to \mathsf{C}]\!]_{\gamma[A/X]}$ to the composition $\mathrm{gpd}[\![\mathsf{C}]\!]_\gamma(i) \circ f \circ \mathrm{gpd}[\![\mathsf{B}]\!]_\gamma(i^{-1})$. □

Now, for any set A in $\mathcal{C}$, let $B \in \mathbf{C}$ be such that $B \cong A$ by way of the isomorphism $i\colon B \to A$. Using the groupoid action defined above, we have $\mathrm{gpd}[\![\mathsf{B}]\!]_\gamma(i)(\pi_B) \in \mathcal{C}[\![\mathsf{B}]\!]_{\gamma[A/X]}$. Similarly, for any algebra $\underline{A}$ in $\mathcal{A}$, let $\underline{B} \in \mathbf{A}$ be such that $\underline{B} \cong^\circ \underline{A}$[3] by way of $j\colon \underline{B} \multimap \underline{A}$. Then we have $\mathrm{gpd}[\![\mathsf{B}]\!]_\gamma(j)(\kappa_{\underline{B}}) \in \mathcal{C}[\![\mathsf{B}]\!]_{\gamma[\underline{A}/\underline{X}]}$.

Lemma A.6 *For* $\pi \in \mathcal{C}[\![\forall X.\, \mathsf{B}]\!]_\gamma$ *and* A *in* $\mathcal{C}$*:*

1. *The value* $\mathrm{gpd}[\![\mathsf{B}]\!]_\gamma(i)(\pi_B)$ *is independent of the choice of* B *and* i*.*
2. *If* $A \in \mathbf{C}$ *then* $\mathrm{gpd}[\![\mathsf{B}]\!]_\gamma(i)(\pi_B) = \pi_A$*.*

Similarly, for $\kappa \in \mathcal{C}[\![\forall \underline{X}.\, \mathsf{B}]\!]_\gamma$ *and* $\underline{A} \in \mathcal{A}$*:*

3. *The value* $\mathrm{gpd}[\![\mathsf{B}]\!]_\gamma(j)(\kappa_{\underline{B}})$ *is independent of the choice of* $\underline{B}$ *and* j*.*
4. *If* $\underline{A} \in \mathbf{A}$ *then* $\mathrm{gpd}[\![\mathsf{B}]\!]_\gamma(j)(\kappa_{\underline{B}}) = \kappa_{\underline{A}}$*.*

Proof. We prove 1. Suppose $i\colon B \to A, i'\colon B' \to A$ are isomorphisms. We must show that $\mathrm{gpd}[\![\mathsf{B}]\!]_\gamma(i)(\pi_B) = \mathrm{gpd}[\![\mathsf{B}]\!]_\gamma(i')(\pi_{B'})$. By the parametricity condition in the definition of $\mathcal{C}[\![\forall X.\, \mathsf{B}]\!]_\gamma$, $\mathcal{C}[\![\mathsf{B}]\!]_{\gamma[\langle i'^{-1} \circ i\rangle/X]}(\pi_B, \pi_{B'})$, which by functoriality of the groupoid action implies $\langle \mathrm{gpd}[\![\mathsf{B}]\!]_\gamma(i')^{-1} \circ \mathrm{gpd}[\![\mathsf{B}]\!]_\gamma(i)\rangle(\pi_B, \pi_{B'})$ as desired. □

[3] We write $\underline{A} \cong^\circ \underline{B}$ if $\underline{A}$ and $\underline{B}$ are isomorphic in $\mathcal{A}$.

The above lemma justifies us writing $\pi(A)$ for $\mathrm{gpd}[\![\mathsf{B}]\!]_\gamma(i)(\pi_B)$ for any set A in $\mathcal{C}$. Similarly, we write $\kappa(\underline{A})$ for $\mathrm{gpd}[\![\mathsf{B}]\!]_\gamma(j)(\kappa_{\underline{B}})$ for any $\underline{A} \in \mathcal{A}$.

Lemma A.7 *1. If $\mathcal{R}[\![\forall X.\ \mathsf{B}]\!]_\rho(\pi, \pi')$ then, for all sets A, B in $\mathcal{C}$ and relations $R \in \mathcal{R}_\mathcal{C}(A, B)$, it holds that $\mathcal{R}[\![\mathsf{B}]\!]_{\rho[R/X]}(\pi(A), \pi'(B))$.*

2. If $\mathcal{R}[\![\forall \underline{X}.\ \mathsf{B}]\!]_\rho(\kappa, \kappa')$ then, for all algebras $\underline{A}, \underline{B}$ in $\mathcal{A}$ and relations $Q \in \mathcal{R}_\mathcal{A}(\underline{A}, \underline{B})$, it holds that $\mathcal{R}[\![\mathsf{B}]\!]_{\rho[Q/\underline{X}]}(\kappa(\underline{A}), \kappa'(\underline{B}))$.

Proof. We just prove item 1 of the lemma, item 2 is proved similarly. Suppose we are given sets A, B in $\mathcal{C}$ and a relation $R \in \mathcal{R}_\mathcal{C}(A, B)$. Then we know that there exists sets $A', B' \in \mathbf{C}$ and isomorphisms $i\colon A' \to A$, $j\colon B' \to B$. By definition, if $\mathcal{R}[\![\forall X.\ \mathsf{B}]\!]_\rho(\pi, \pi')$ then $\mathcal{R}[\![\mathsf{B}]\!]_{\rho[(i,j)^{-1}R/X]}(\pi_{A'}, \pi'_{B'})$. A simple inductive check shows that $\mathcal{R}[\![\mathsf{B}]\!]_{\rho[(i,j)^{-1}R/X]} = (\mathrm{gpd}[\![\mathsf{B}]\!]_{\rho_1}(i), \mathrm{gpd}[\![\mathsf{B}]\!]_{\rho_2}(j))^{-1}\mathcal{R}[\![\mathsf{B}]\!]_{\rho[R/X]}$, so $(\pi(A), \pi'(B)) = (\mathrm{gpd}[\![\mathsf{B}]\!]_{\rho_1}(i^{-1})(\pi_{A'}), \mathrm{gpd}[\![\mathsf{B}]\!]_{\rho_2}(j^{-1})(\pi'_{A'}))$ are in $\mathcal{R}[\![\mathsf{B}]\!]_{\rho[R/X]}(\pi(A), \pi'(B))$. □

Taken together, Lemmas A.6 and A.7 justify that Figure 6 can indeed be viewed as defining polymorphic types in terms of the large products used in Figure 3.

Proof of Proposition 4.3 (sketch). The three statements of the proposition must be proved simultaniously by structural induction on t. Most of the cases are standard and we just show a few.

We prove the homomorphism property in the case of application of a polymorphic term $t\colon \forall X.\ \underline{B}$ to a value type A. By definition $\mathcal{A}[\![\forall X.\ \underline{B}]\!]_\gamma$ is a subobject of a product taken in $\mathcal{A}$, and $[\![t(\mathsf{A})]\!]_\gamma$ is defined to be

$$\mathrm{gpd}[\![\underline{B}]\!]_\gamma(i)([\![t]\!]_\gamma)_A$$

for any $A \in \mathbf{C}$, and isomorphism $i\colon A \to \mathcal{C}[\![\mathsf{A}]\!]_\gamma$. But since the subobject and the product are taken in the category $\mathcal{A}$, the inclusion and projection in question are homomorphisms, and by Lemma A.5 $\mathrm{gpd}[\![\underline{B}]\!]_\gamma(i)$ is also a homomorphism proving the inductive case of the homomorphism property.

The homomorphism property in the case of function application $t(s)$ for $t\colon \underline{\mathsf{B}} \multimap \underline{\mathsf{C}}$ follows from well definedness: by induction hypothesis $[\![t]\!]_\gamma \in \mathcal{A}[\![\underline{\mathsf{B}} \multimap \underline{\mathsf{C}}]\!]_\gamma$ and so is a homomorphism, so if $d \in \mathcal{C}[\![\underline{\mathsf{A}}]\!]_\gamma \mapsto [\![s]\!]_{\gamma[d/x]}$ is a homomorphism so is $d \in \mathcal{C}[\![\underline{\mathsf{A}}]\!]_\gamma \mapsto [\![t]\!]_\gamma([\![s]\!]_{\gamma[d/x]})$. Likewise well definedness in the case of linear lambda abstraction: $\lambda^\circ x\colon \underline{\mathsf{A}}.\ t$ follows from the homomorphism property for t.

We show well definedness in one of the cases of polymorphic lambda abstraction: $\Lambda X.\ t\colon \forall X.\ \mathsf{B}$. Here we must show that $\{[\![t]\!]_{\gamma[A/X]}\}_{A \in \mathbf{C}}$ satisfies the parametricity condition in the definition of $\mathcal{C}[\![\forall X.\ \mathsf{B}]\!]_\gamma$: for all $A, B \in \mathbf{C}$ and all relations $R \in \mathcal{R}_\mathcal{C}(A, B)$,

$$\mathcal{R}[\![\mathsf{B}]\!]_{\Delta_\gamma[R/X]}([\![t]\!]_{\gamma[A/X]}, [\![t]\!]_{\gamma[B/X]})$$

This follows from the relational invariance property for t, as assumed in the induction hypothesis, since $\mathcal{R}[\![\Gamma]\!]_{\Delta_\gamma}(\gamma, \gamma)$ holds by the identity extension lemma. Likewise, the relational invariance property in the case of type application of polymorphic terms follows from well definedness using Lemma A.7. □

A.4 Section 5

Proof of Proposition 5.3. We first show that if $(x, y) \in R$ then $(\eta_A(x), \eta_B(y)) \in {!R}$. So suppose we are given $\underline{A}, \underline{B} \in \mathcal{A}$ and $Q \in \mathcal{R}_\mathcal{A}(\underline{A}, \underline{B})$. We must show that if $f\colon A \to U\underline{A}, g\colon B \to U\underline{B}$ satisfy $(R \to Q)(f, g)$ then $(\eta_A(x)(\underline{A})(f), \eta_B(y)(\underline{B})(g)$. But this follows from definition of $(R \to Q)$ since $(\eta_A(x)(\underline{A})(f), \eta_B(y)(\underline{B})(g) = (f(x), g(y))$.

Now, suppose $Q \in \mathcal{R}_\mathcal{A}(!A, !B)$ and for all $(x, y) \in R$ we have $Q(\eta_A(x), \eta_B(y))$, or in other words $(R \to Q)(\eta_A, \eta_B)$. We must show that $!R \subseteq Q$. So suppose $(z, z') \in {!R}$. By definition of $!R$ using $(R \to Q)(\eta_A, \eta_B)$ we have

$$Q(z(!A)(\eta_A), z'(!B)(\eta_B)).$$

But by definition $z(!A)(\eta_A) = \text{let } !x \text{ be } z \text{ in } !x$ which by Lemma 5.1 is equal to z. Likewise $z'(!B)(\eta_B) = z'$ proving $Q(z, z')$. □

Here is a different description of $!R$, that we shall need below.

Lemma A.8 *If $A, B \in \mathcal{C}$, $\underline{A}, \underline{B} \in \mathcal{A}$, $R \in \mathcal{R}_\mathcal{C}(A, B)$, $Q \in \mathcal{R}_\mathcal{A}(\underline{A}, \underline{B})$ and $f\colon !A \multimap \underline{A}, g\colon !B \multimap \underline{B}$, then $(!R \multimap Q)(f, g)$ iff $(R \to Q)(f \circ \eta_A, g \circ \eta_B)$.*

Proof. The "only if" direction is simply because $(R \to {!R})(\eta_A, \eta_B)$. On the other hand, if $(R \to Q)(f \circ \eta_A, g \circ \eta_B)$ then $(f, g)^{-1}Q$ is an admissible relation containing all elements of the form $(\eta_A(x), \eta_B(y))$ for which $R(x, y)$ hold, and so by Proposition 5.3 must contain $!R$ proving $(!R \multimap Q)(f, g)$. □

A.5 Section 6

Proof of Theorem 6.1. The equivalence of statements 2 and 3 is immediate from (1), because $TA = !A$. So we establish the equivalence of 1 and 3. Suppose that θ is a natural transformation from $(U(-))^A$ to U. We show that the mapping $\underline{A} \in \mathcal{A} \mapsto \lambda f\colon A \to U\underline{A}.\ \theta_{\underline{A}}(f)$ is an element of $\forall \underline{X}.\ (A \to \underline{X}) \to \underline{X}$. Suppose $\underline{A}, \underline{B} \in \mathcal{A}$ and

$Q \in \mathcal{R}_{\mathcal{A}}(\underline{A}, \underline{B})$. We must show that $(\Delta_A \to Q)\,(f, g)$ implies $Q(\theta_{\underline{A}}(f), \theta_{\underline{B}}(g))$. The projections $\pi_1 \colon Q \to U\underline{A}$ and $\pi_2 \colon Q \to U\underline{B}$ are homomorphisms from the algebra carried by Q to $\underline{A}$ and $\underline{B}$ respectively. By applying naturality to the corresponding maps in $\mathcal{A}$, the two squares below commute.

$$\begin{tikzcd}
(U\underline{A})^A \arrow[d, "\theta_{\underline{A}}"'] & Q^A \arrow[l, "(\pi_1)^A"'] \arrow[r, "(\pi_2)^A"] \arrow[d, "\theta_C"] & U\underline{B}^A \arrow[d, "\theta_{\underline{B}}"] \\
U\underline{A} & Q \arrow[l, "\pi_1"] \arrow[r, "\pi_2"'] & U\underline{B}
\end{tikzcd}$$

But this says that, for any f, g with $Q(f(x), g(x))$ for all $x \in A$, it holds that $Q(\theta_{\underline{A}}(f), \theta_{\underline{B}}(g))$, which is what we needed to show. For the converse direction, consider any $\kappa \in \forall \underline{X}.\, (A \to \underline{X}) \to \underline{X}$. Then $\theta_{\underline{A}}(f) = \kappa(\underline{A})(f)$ is the corresponding algebraic operation. Verifying naturality is a routine use of graphs of homorphisms cf. [18]. It is obvious that the two constructions are mutually inverse. $\square$

Lemma A.9 *Suppose $\underline{A}, \underline{B} \in \mathcal{A}$ and $R \subseteq U\underline{A} \times U\underline{B}$ is any subset. Then there exists a smallest admissible relation $R^\circ \in \mathcal{R}_{\mathcal{A}}(\underline{A}, \underline{B})$ containing R. Moreover,*

1. *If $R \subseteq U\underline{A} \times U\underline{B}$ and $R' \subseteq U\underline{A}' \times U\underline{B}'$ then $(R \times R')^\circ = R^\circ \times R'^\circ$.*
2. *If $A, B \in \mathcal{C}$ and $R \in \mathcal{R}_{\mathcal{C}}(A, B)$ then $\mathrm{im}(TR \to TA \times TB)^\circ = {!}R$. Where $\mathrm{im}(TR \to TA \times TB)$ is the image of map obtained by applying the functor T to the span corresponding to R.*

Proof. The relation R° is the intersection of all admissible relations containing R. We show property 2. Since $(R \to \mathrm{im}(TR \to TA \times TB)^\circ)(\eta_A, \eta_B)$, by Lemma A.8 ${!}R \subseteq \mathrm{im}(TR \to TA \times TB)^\circ$. For the other inclusion notice that since $(R \to {!}R)(\eta_A, \eta_B)$, naturality of the correspondence given by Theorem 5.2 implies the existence of a map h making the diagram

$$\begin{tikzcd}
FA \arrow[d, -o, "\mathrm{id}_B"'] & FR \arrow[l, -o, "F\pi_1"'] \arrow[r, -o, "F\pi_2"] \arrow[d, -o, "h"] & FB \arrow[d, -o, "\mathrm{id}_B"] \\
FA & {!}R \arrow[l, -o] \arrow[r, -o] & FB
\end{tikzcd}$$

commute. This proves $\mathrm{im}(TR \to TA \times TB) \subseteq {!}R$. Since ${!}R$ is admissible we get $\mathrm{im}(TR \to TA \times TB)^\circ \subseteq {!}R$. $\square$

Proof of Theorem 6.2. An element of $\forall X.\, (n \to {!}X) \multimap {!}X$ gives for each $A \in \mathcal{C}$ a map $(FA)^n \multimap FA$, and the naturality square for this family follows from the parametricity condition satisfied by elements of polymorphic type, applied to the graph of a function. The difficult part of this proof is to show that natural transformations satisfy the parametricity condition and thus define elements of $\forall X.\, (n \to {!}X) \multimap {!}X$.

So suppose $(f_A \colon (FA)^n \multimap FA)_{A \in \mathcal{C}}$ is a natural transformation, and $A, B \in \mathcal{C}$ and $R \in \mathcal{R}_{\mathcal{C}}(A, B)$. We must show that $(({!}R)^n \multimap {!}R)(f_A, f_B)$. Naturality applied to the span $A \leftarrow R \to B$ gives us commutativity of

$$\begin{tikzcd}
(FA)^n \arrow[d, -o, "f_B"'] & (FR)^n \arrow[l, -o, "(F\pi_1)^n"'] \arrow[r, -o, "(F\pi_2)^n"] \arrow[d, -o, "f_R"] & (FB)^n \arrow[d, -o, "f_B"] \\
FA & FR \arrow[l, -o, "F\pi_1"'] \arrow[r, -o, "F\pi_2"] & FB
\end{tikzcd}$$

Since f_A and f_B are homomorphisms, this implies

$$(\mathrm{im}((TR)^n)^\circ \multimap \mathrm{im}(TR)^\circ)(f_A, f_B)$$

which by Lemma A.9 implies

$$(({!}R)^n \multimap {!}R)(f_A, f_B)$$

as desired. $\square$