

## FUNCTORIAL POLYMORPHISM

E.S. BAINBRIDGE

*Department of Mathematics, University of Ottawa, 585 King Edward, Ottawa, Ontario, Canada  
K1N 6N5*

P.J. FREYD\* and A. SCEDROV\*\*

*Department of Mathematics, University of Pennsylvania, Philadelphia, PA 19104-6395, USA*

P.J. SCOTT\*\*\*

*Department of Mathematics, University of Ottawa, 585 King Edward, Ottawa, Ontario, Canada  
K1N 6N5*

### Contents

0. Introduction .....	35
1. The functorial calculus .....	38
2. Realizable endofunctors on <i>PER</i> .....	43
3. Interpreting the syntax .....	47
4. Reynolds parametricity in pers .....	48
Appendix .....	59
Acknowledgment .....	62
References .....	62

### 0. Introduction

In the past several years types have become an important component of programming language design. They provide a logical framework to ensure that programs meet given specifications, support a partial correctness or verification mechanism, enhance software maintenance, and encourage the systematic building of complex modules from simpler ones. These features are crucial in large-scale programming projects requiring coordination among many teams of programmers.

Many recently developed programming languages have more sophisticated typing mechanisms than the familiar Algol/Pascal family. For example ML-like languages [23, 40, 62, 15], as well as such languages as Ada [3] and Clu [35], feature aspects of *polymorphic* or generic data types which allow the programmer great flexibility and abstraction.

\* Partially supported by grants from the U.S. Office of Naval Research and the U.S. National Science Foundation.

\*\* Partially supported by the U.S. Office of Naval Research Grant N00014-88-K-0635, by the U.S. National Science Foundation Grant CCR-8705596, and by the 1987 Young Faculty Award from the Natural Sciences Association of the University of Pennsylvania.

\*\*\* Supported by an operating grant from the Natural Sciences and Engineering Research Council of Canada and member of a team grant from FCAR (Quebec).

Various notions of polymorphism were first introduced into computer science by Strachey [60] (cf. also [41, 48, 10]). Among the most influential was the notion of *parametric* polymorphism. Intuitively, a parametric polymorphic function is one that *has a uniformly given algorithm in all types*. An important example is Strachey's map-list.

**Example.** Consider a function  $f$  whose argument is of type  $\alpha$  and whose result is of type  $\beta$ , so the type of  $f$  is  $\alpha \Rightarrow \beta$ . Let  $L$  be a list of elements of type  $\alpha$ . We say that  $L$  is of type  $\alpha$ -list. Now consider the following function *map*: apply  $f$  to the individual entries of  $L$ , then make a list of the results. Thus  $\text{map}(f)(L)$  is a list of elements of type  $\beta$ , so *map* is of type  $(\alpha \Rightarrow \beta) \Rightarrow (\alpha\text{-list} \Rightarrow \beta\text{-list})$ . Note that no specific properties of the types  $\alpha$  and  $\beta$  are used; in fact, we might as well suppose they are variables.

Studies of typed functional languages often employ various extensions of typed lambda calculi (see, e.g. [34, 45, 47, 42, 26]). Studying such formal calculi provides many mathematical insights into the structure of programming languages. In the case of polymorphism, a formal calculus of variable types (higher-order lambda calculus) was developed by Girard [20, 21] as a higher-order extension of the Curry-Howard propositions-as-types paradigm in mathematical logic (types correspond to formulae, terms to deductions or proofs: cf. [24, 38]). Reynolds [46] independently discovered the second order fragment of this calculus and proposed it as a syntax for Strachey's parametric polymorphism. Several powerful extensions of the Girard-Reynolds calculus have been studied and implemented, most notably the Coquand-Huet Calculus of Constructions [12, 25]. Recent mathematical advances in type theory and lambda calculi have already influenced program-language design (in addition to the calculus of constructions, cf. Nuprl [11], Quest [9], Forsythe [50]).

Girard [21] analyzed the syntax and metatheory of higher-order lambda calculi. Many papers have explored the difficult semantical problems intrinsic to such theories (for a recent survey, see [53]). For example, Reynolds [48] suggested that second order lambda calculus might have a set theoretic interpretation. Reynolds [49] (cf. also [51]) then showed that in fact this calculus *cannot* have a non-trivial set model. More recently, Pitts [44] showed that if we use *intuitionistic* (or constructive) set theory, there are many models, enough for a completeness theorem. Other model-theoretic studies of polymorphic lambda calculi are given in [7, 58, 39].

The starting point for this paper was the simple question: what do types and terms of polymorphic lambda calculus refer to, given that there are no (classical) Set-models [49, 51]. The polymorphic identity term gives a clue. Consider  $\Lambda\alpha(\lambda x:\alpha.x)$  of type  $\forall\alpha(\alpha \Rightarrow \alpha)$ . Then  $\Lambda\alpha(\lambda x:\alpha.x)[A] = \lambda x:A.x$ . That is, this term, when applied or instantiated at a type  $A$  gives the identity function on  $A$ . Thus  $\Lambda\alpha(\lambda x:\alpha.x)$  is a *type-indexed family of (identity) functions*.

Therefore the first attempt is to take  $\forall\alpha(\alpha \Rightarrow \alpha)$  as the collection of all indexed families of functions from  $\alpha$  to  $\alpha$ , which amounts to taking the product, to wit

$\prod_A (A \Rightarrow A)$  over all types  $A$ . As argued in [48], this product is too big. In Strachey's terminology, this product also contains *ad hoc* elements which are not intended in the notion of *parametric* polymorphism. Reynolds [48] proposes to distinguish *parametric* elements of such products by means of an invariance condition with respect to certain relations induced by polymorphic types. In [48] this invariance requirement was discussed in the context of an attempted set-theoretic model, which was subsequently shown to be impossible [49, 51].

Our point of departure is to insist on certain naturality (or uniformity) conditions on these indexed families. Therefore universal type abstraction will not be interpreted simply as a product, but rather as that part of the product consisting only of those families that satisfy naturality conditions intended to reflect parametricity. So, our next approximation is that types are functors, and terms are natural transformations between types, all defined over some cartesian closed category (ccc)  $\mathcal{C}$  of "ground" types. Note that the constant functors really play the role of "ground" types. The function space type  $\alpha \Rightarrow \beta$  corresponds to the internal hom functor  $(\ ) \Rightarrow (\ ) : \mathcal{C}^\circ \times \mathcal{C} \rightarrow \mathcal{C}$  which is contravariant in its first argument and covariant in its second argument. Alas,  $\alpha \Rightarrow \alpha$  would have to be both contravariant and covariant in  $\alpha$ , i.e.  $\alpha \Rightarrow \alpha$  is just not a functor.

There are at least two ways of resolving this problem. One approach [54, 59, 22] is to move to a category of retract pairs, which serves to obliterate the difference between covariant and contravariant. Naturality conditions so obtained are rather weak and thus the interpretation of universal type abstraction given by this approach still allows *ad hoc* elements.

This problem, in fact, is not a new one. It was encountered several decades ago in algebraic topology. Moving to retract pairs would not have been useful in that context. Another approach based on a suitably generalized notion of "multivariant" natural transformation was developed. A calculus of these generalized natural transformations started with Yoneda [63] and later Dubuc and Street [16] (cf. the discussion in [37, pp. 214–228]). Indeed, the observation that the calculus of generalized natural transformations closely resembles the second-order  $\beta$ - $\eta$  rules of polymorphic lambda calculus led to our whole project. Moreover, as we are now discovering, these notions of naturality capture certain semantic aspects of parametricity; in particular, *they permit a semantically sound and systematic approach to adding new equations to the syntax.*

In this paper we consider two semantic approximations to Strachey's notion of parametricity. The first one, discussed in sections 1–3, is based on the above-mentioned calculus of generalized natural transformations. The second one, discussed in Section 4, is based on a version of the Reynolds invariance condition referred to earlier.

Basic definitions and examples of the calculus of generalized natural transformations are given in Section 1. In particular, a semantic analog of universal type abstraction is discussed in Section 1.1.

In Section 2 we study the calculus of generalized natural transformations, in the context of partial equivalence relations (*pers*) semantics (see [41, 5]). Remarkably,

in this context, there is an extensive class of generalized natural transformations (Theorem 2.9). This yields a setting for semantics of polymorphism in which elements of universal types are only those families which belong to this class of generalized natural transformations (Theorem 2.11). The corresponding interpretation of the syntax of polymorphic lambda calculus is presented in Section 3.

In Section 4 we turn to the Reynolds parametricity condition (see above). We develop a consistent framework for this condition in the context of **per** semantics. This yields an interpretation of polymorphism with a built-in requirement of the parametricity of elements of universal types (Section 4.2). In this interpretation, for any second-order definable covariant functor  $T$ , the universal type  $\forall \alpha ((T\alpha \Rightarrow \alpha) \Rightarrow \alpha)$  must be the initial  $T$ -algebra.

**Notation.** Composition of arrows in a category will be denoted two ways: if  $f: A \rightarrow B$ ,  $g: B \rightarrow C$ , their diagrammatic composite (composite in order of execution) is denoted  $f;g: A \rightarrow C$ , while the equivalent composite in usual mathematical notation is denoted  $g \circ f: A \rightarrow C$ . If  $\mathcal{C}^n$  is a product category, we use **bold face** letters to denote vectors of objects or arrows. For example, in  $\mathcal{C}^n$ ,  $\mathbf{f};\mathbf{g}: \mathbf{A} \rightarrow \mathbf{C}$  denotes the vector of arrows whose  $i$ th component is  $f_i;g_i: A_i \rightarrow C_i$ .  $\mathcal{C}^{\text{op}}$  denotes the opposite category  $\mathcal{C}^{\text{op}}$ .

In a cartesian closed category, we shall denote exponentiation either by the categorical notation  $B^A$  or sometimes the logicians' notation  $A \Rightarrow B$ , whichever is appropriate.  $\mathbf{N}$  denotes the natural numbers  $\{0, 1, 2, \dots\}$ .

## 1. The functorial calculus

In what follows,  $\mathcal{C}$  is a cartesian closed category. As mentioned in the Introduction, the general plan is to interpret types as ranging over some class of multivariate  $n$ -ary functors  $F: (\mathcal{C}^{\text{op}})^n \times \mathcal{C}^n \rightarrow \mathcal{C}$  and to interpret terms as ranging over some appropriate class of "multivariate" natural transformations.

**1.0. Definition.** A *dinatural transformation* between two functors  $F, G: (\mathcal{C}^{\text{op}})^n \times \mathcal{C}^n \rightarrow \mathcal{C}$  is a family of morphisms  $u = \{u_A: FAA \rightarrow GAA \mid A \in \mathcal{C}^n\}$  satisfying the following condition: for any vector of morphisms  $\mathbf{f}: \mathbf{A} \rightarrow \mathbf{B}$ ,

$$\begin{array}{ccccc}
 & & FAA & \xrightarrow{u_A} & GAA \\
 & \nearrow^{FfA} & & & \searrow^{GfA} \\
 FBA & & & & GAB \\
 & \searrow^{FfB} & & & \nearrow^{GfB} \\
 & & FBB & \xrightarrow{u_B} & GBB
 \end{array} \tag{*}$$

In many examples  $u$  is given by a "uniform algorithm" which is the "same" for each object  $A$ . This is illustrated in the examples and models below. We sometimes speak of (\*) as the *hexagon property*. We use the notation  $u: F \rightarrow G$  to denote that  $u$  is a dinatural transformation from  $F$  to  $G$ .

We mention three special cases and then consider some examples.

(a) Suppose  $F$  and  $G$  are covariant:  $\mathcal{C} \rightarrow \mathcal{C}$ , thought of as functors  $\mathcal{C} \times \mathcal{C} \rightarrow \mathcal{C}$ , dummy in the first argument. Then the definition of dinatural transformation  $u$  reduces to that of an ordinary natural transformation since the two oblique arrows  $FBA \rightarrow FAA$  and  $GBB \rightarrow GAB$  are identities.

(b) A less familiar example is when one functor, say  $F$ , is covariant and the other functor  $G$  is contravariant. Then a dinatural transformation from  $F$  to  $G$  is a family  $u$  satisfying the following diagram for any  $f: A \rightarrow B$ ,

$$\begin{array}{ccc} FA & \xrightarrow{u_A} & GA \\ Ff \downarrow & & \uparrow Gf \\ FB & \xrightarrow{u_B} & GB \end{array}$$

(c) Suppose one of  $F$  or  $G$  is constant: say  $F$  is  $K_D$ , the constant functor whose value (on objects) is always  $D$ . Then the notion of dinatural transformation reduces to the case of a family  $u = \{u_A: D \rightarrow GAA\}$  satisfying the following wedge condition (cf. [37, 63]): for any  $f: A \rightarrow B$ ,

$$\begin{array}{ccc} & GAA & \\ u_A \nearrow & & \searrow GAf \\ D & & GAB \\ u_B \searrow & & \nearrow GfB \\ & GBB & \end{array}$$

**1.1. Examples.** The following examples make sense in SETS or more generally in any cartesian closed category  $\mathcal{C}$ .

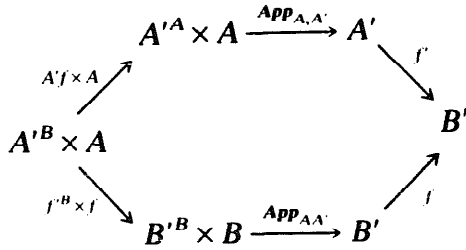
(i) *Polymorphic identity*: let  $K_1$  be the constant functor with value the terminal object  $\mathbf{1}$  and  $(\ )^{(\ )}$  the internal function space (hom) functor. Consider the uniform family  $u: K_1 \rightarrow (\ )^{(\ )}$ , where  $u_A: \mathbf{1} \rightarrow A^A$  is given by  $\lambda x: A.x$  (known in category theory as the "name" of the identity on  $A$ ). The dinaturality condition reduces to: for every arrow  $f: A \rightarrow B$ ,

$$\begin{array}{ccc} & A^A & \\ u_A \nearrow & & \searrow f^A \\ \mathbf{1} & & B^A \\ u_B \searrow & & \nearrow B^f \\ & B^B & \end{array}$$

which is equivalent to saying that  $f \circ \text{id}_A = \text{id}_B \circ f$ , which is certainly true. Note also that  $u$  is given uniformly at any type slot  $(\ )$  by  $\lambda x: (\ ).x$ .

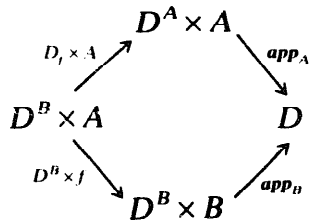
(ii) *Uniform Church numeral n*: for each object  $A$  in  $\mathcal{C}$ , consider the family of  $A$ -indexed arrows  $n = \{n_A : A^A \rightarrow A^A \mid A \in \mathcal{C}\}$ , where  $n_A$  maps  $h$  to the  $n$ -fold composition of  $h$  with itself,  $h^n$ . Then  $n$  determines a dinatural transformation  $( )^{(\cdot)} \rightarrow ( )^{(\cdot)}$ . Indeed, the reader can compute that the dinaturality condition says that for any  $f : B^A, g : A^B, f \circ (g \circ f)^n = (f \circ g)^n \circ f$ , which is an instance of associativity. Again note the “uniformity” in the definition of the algorithm for (each component of)  $n$ .

(iii) *Application*: Consider the family  $\mathbf{App}$ , with components  $\mathbf{App}_{A,A'} : (A')^A \times A \rightarrow A'$  given by application (or evaluation) in the ccc  $\mathcal{C}$ , for each pair of objects  $A, A'$ . Then for all  $f : A \rightarrow B$  and  $f' : A' \rightarrow B'$ , the following hexagonal diagram commutes:



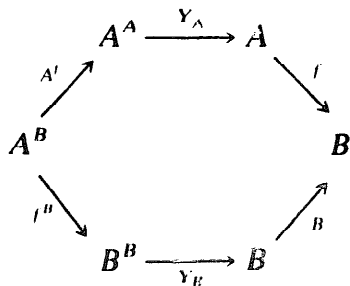
This says if  $g : A'^B, f'((g \circ f)(a)) = (f' \circ g)(f(a))$ .  $\mathbf{App}$  is a dinatural transformation (between functors constructed using the operations in Section 1.5; see Appendix A.6).

As a special case of the above example, let  $D$  be a fixed object in  $\mathcal{C}$ . Consider the dinatural transformation  $\mathbf{app} : D^{(\cdot)} \times ( ) \rightarrow K_D$ , where  $\mathbf{app}_A : D^A \times A \rightarrow D$  is application in the ccc  $\mathcal{C}$ . So at any type slot  $( )$ ,  $\mathbf{app}_{( )}$  has a uniform algorithm (e.g. an appropriate  $\lambda$ -term). It is easily checked that the hexagon condition reduces to the following: for any  $f : A \rightarrow B$ ,



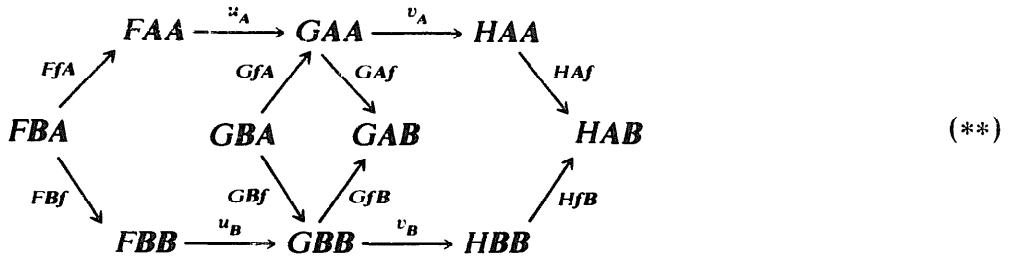
whose commutativity reduces to  $(g \circ f)(a) = g(f(a))$ , for any  $a : A$ , which is true in any ccc.

(iv) *Fixed point combinators*: Suppose there were to exist a dinatural transformation  $Y : ( )^{(\cdot)} \rightarrow \text{id}$ , where the functor  $\text{id}$  denotes the identity functor. So  $Y = \{Y_A : A^A \rightarrow A \mid A \in \mathcal{C}\}$  is a family satisfying the following hexagon condition: for any  $f : A \rightarrow B$ ,



This means (using set-theoretic notation): if  $g: A^B$ ,  $f(Y_A(g \circ f)) = Y_B(f \circ g)$ . In particular, setting  $A \equiv B$  and  $g = \lambda x: A.x$  (the identity on  $A$ ), we see  $Y_A$  must be a fixed point combinator at each type  $A$  (cf. also Appendix A.2).

The most perverse aspect of the calculus of dinatural transformations is the failure of composition. One attempts to compose two dinatural transformations  $u: F \rightarrow G$  and  $v: G \rightarrow H$  by horizontally “merging” the two hexagons; i.e. define their composite  $u, v$  by the formula  $(u; v)_A = u_A; v_A$ . Then for any  $f: A \rightarrow B$ , consider



While both hexagons individually commute, the outer hexagon need not commute. We now give two examples.

**Example.** In the category **SET**, consider the dinatural transformation  $v: ( )^{(\cdot)} \rightarrow K_{\mathbf{BOOLE}}$ , where  $\mathbf{BOOLE} = \{0, 1\}$  and the map  $v_A: A^A \rightarrow \mathbf{BOOLE}$ , where  $v_A$  is 0 or 1 depending upon whether the number of fixed points of its argument is even or odd, understanding  $\infty$  as even. Consider the polymorphic identity  $u: K_1 \rightarrow ( )^{(\cdot)}$  (see Example 1.1(i)). The map  $(u; v)_A: 1 \rightarrow \mathbf{BOOLE}$  depends on  $A$  (it is not constant in  $A$ ); so  $u; v$  cannot be a dinatural transformation between constant functors.

**Example.** Let  $\mathcal{C}$  be a ccc. Take any dinatural transformation  $Y: ( )^{(\cdot)} \rightarrow ( )$  (cf. Example 1.1(iv)). If we were able to compose  $Y$  with the polymorphic identity  $K_1 \rightarrow ( )^{(\cdot)}$ , then the category  $\mathcal{C}$  would be degenerate (i.e. given any ordered pair of objects, there is a *unique* map from the first to the second). See Appendix A.4.

**1.2. Fact.** *If the middle diamond in (\*\*) is a pullback, then we can in fact compose the dinatural transformations  $u: F \rightarrow G$  and  $v: G \rightarrow H$  above. For in this case, there exists a horizontal arrow  $FBA \rightarrow GBA$  making everything in sight commute.*

The problem of composing dinatural and other classes of generalized natural transformations has been examined by various authors (e.g. [17, 31]). At a general level these problems are quite intricate and only partial solutions are known. In Sections 2 and 3, these obstacles to compositionality will be resolved for certain large classes of multivariant functors and dinatural transformations intrinsic to PER and HEO-like models (cf. [41, 5, 36]). We show that this includes at least those functors and dinatural transformations definable in second-order polymorphic lambda calculus.

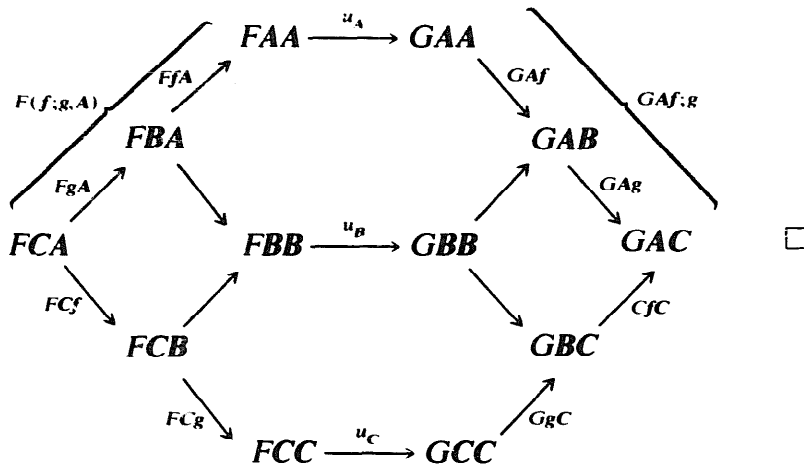
For the remainder of this section we stay at a general level: some class of multivariant functors with some (not necessarily compositional) dinatural transformations between them. But the reader should bear in mind that we can always

specialize the entire framework to some nice model where, among other things, compositionality holds.

We can say a bit more about compositionality at the general level. Until further notice, let us fix functors  $F, G, H: (\mathcal{C}^0)^n \times \mathcal{C}^n \rightarrow \mathcal{C}$ . Let  $u, v$  denote families of morphisms  $u_A: FAA \rightarrow GAA, v_A: GAA \rightarrow HAA, A \in \mathcal{C}$ , not necessarily dinatural. Let  $D_u$  be the set of  $f$  for which the family  $u$  is dinatural; that is  $f \in D_u$  iff the hexagon (\*) in Definition 1.0 commutes.  $D_v$  and  $D_{u,v}$  are defined similarly.

**1.3. Proposition (Vertical merging).**  $D_u$  is a subcategory.

**Proof.**  $D_u$  trivially contains identity arrows. Given  $f: A \rightarrow B$  and  $g: B \rightarrow C$  in  $D_u$ , stare at the following diagram, using functoriality of  $F$  and  $G$  on the oblique outer edges:



**1.4. Proposition (Horizontal merging with respect to isomorphisms).** Any isomorphism in  $D_u$  and  $D_v$  is necessarily in  $D_{u,v}$ .

**Proof.** Suppose  $f$  is an isomorphism (iso) in both  $D_u$  and  $D_v$ . Observe that the middle diamond in (\*\*) has each side the  $G$ -image of an iso. Since functors preserve isos, we obtain a diamond whose sides are all isos, and such a diagram is of course a pullback; now use Fact 1.2.  $\square$

**1.5. Some functors**

We can build new functors from old by various operations. For the record, given two functors  $F$  and  $G$ , their *product* is constructed pointwise; that is, on objects,  $(F \times G)(A, B) = F(A, B) \times G(A, B)$ , while their *exponential* or *function space*  $G^F$  is the functor  $G^F(A, B) = G(A, B)^{F(B,A)}$ . It is a remarkable fact that  $G^F$  is the categorically defined exponentiation in certain very special situations, an example of which is described in Section 2 (see also Appendix A.6, and the recent work in [18]).



One of the most ubiquitous constructions in applied category theory given first by Yoneda [63] (cf. [37, pp.218-224]) plays a critical role in our semantics for polymorphism. Given a functor  $G$ , we seek a universal wedge into  $G$ , that is, an object  $E$  and a dinatural transformation  $K_E \rightarrow G$ , universal for all such dinatural transformations. That is, for any dinatural transformation  $K_D \rightarrow G$ , there is a unique triangle

$$\begin{array}{ccc} K_D & & \\ \downarrow \text{dashed} & \searrow & \\ K_E & \longrightarrow & G. \end{array}$$

$E$ , when it exists, is called the *end* of  $G$ , and is denoted by  $\int_A GAA$ . (This notation happily displays  $A$  as a bound variable; indeed,  $A$  could even be a vector of “variables”.) We note that there may be other variables in the expression  $G$  then those specifically noted, and the universal property guarantees that  $\int_A GAA$  will be a functorial in these other variables.

One may think of  $\int_A GAA$  as a *subset* of the product  $\prod_A GAA$ ; namely,

$$\int_A GAA = \{g \in \prod_A GAA \mid G(A, f)(g_A) = G(f, B)(g_B) \text{ for all } f: A \rightarrow B \in \mathcal{C}\}.$$

Note however that the product is taken over the class of all objects of  $\mathcal{C}$ . In the case that  $G$  is covariant, the end is the usual notion of the limit of the functor  $G$  [37, p. 68].

**1.6. Proposition.** *Maps  $1 \rightarrow \int_A FAA \Rightarrow GAA$  are in bijective correspondence with dinatural transformations from  $F$  to  $G$  (and this correspondence is natural in any other variables).*

**Proof.** The proof is a matter of identity checking.  $\square$

**2. Realizable endofunctors on PER**

A *partial equivalence relation* (per) on a set  $A$  is a symmetric, transitive relation  $R$  on  $A$ . Hence  $R$  is an equivalence relation on  $\text{dom}_R = \{a \in A \mid a R a\} \subseteq A$ . One may think of  $R$  as partitioning  $\text{dom}_R$  into disjoint classes.

Let  $\text{per}(A)$  denote the set of pers on  $A$ . We first examine the case  $A = \mathbb{N}$ , the natural numbers. Generalizations are considered below.

**2.1. Definition** (*the category of pers*).  $\text{PER}(\mathbb{N})$  is the following category: its *objects* are  $\text{per}(\mathbb{N})$ , the pers on  $\mathbb{N}$ . Given  $E, E' \in \text{PER}(\mathbb{N})$ , a *morphism* from  $E$  to  $E'$  is named by a partial recursive function  $f$  if  $f$  induces a map of quotients  $\text{dom}_E / E \rightarrow \text{dom}_{E'} / E'$ : that is,  $f$  names a morphism if, whenever  $n E m$ , then  $f(n), f(m)$  are defined and  $f(n) E' f(m)$ . We write  $f(n) \downarrow$  for “ $f(n)$  is defined”.

Two morphisms  $f, g: E \rightarrow E'$  are *equal* if the induced maps of quotients are equal, that is  $\forall m, n \in \text{dom}_E, n E m$  implies  $f(n) \downarrow, g(m) \downarrow$  and  $f(n) E' g(m)$ . So  $f$  and  $g$  name the same morphism if  $f(n) E' g(n)$  for all  $n \in \text{dom}_E$ .

We use the notation  $nm$  to denote that the  $n$ th partial recursive function (in some standard enumeration) is applied to  $m$ .

**2.2. Proposition.** *PER(N) is a cartesian closed category.*

**Proof.** *PER(N)* is easily verified to be a category.  $\mathbf{1}$  is any per with a unique equivalence class.  $A \times B$  may be constructed as  $\langle m, n \rangle (A \times B) \langle p, q \rangle$  if  $m A p$  and  $n B q$ , where  $\langle \cdot, \cdot \rangle: \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$  is a chosen recursive bijection.  $B^A$  is the relation on  $\mathbb{N}$  such that  $m (B^A) n$  iff  $m$  and  $n$  are codes of equal morphisms  $A \rightarrow B$ . (in other words,  $m (B^A) n$  iff for all  $i, j \ i A j$  implies  $mi B nj$ ). The fact that this is a ccc uses some elementary recursion theory (cf. Remark 2.3). For example, the fundamental bijection between  $A \times B \rightarrow C$  and  $A \rightarrow C^B$  uses the *S-m-n* theorem.

**2.3. Remark.** Definition 2.1 and Proposition 2.2 make sense if we replace  $\mathbb{N}$  by any partial combinatory algebra  $A = (A, \bullet, S, K)$  (e.g. [2, 33, 5]).

From now on, we restrict ourselves to the category *PER(N)*, denoted **PER**. However, all subsequent discussions apply equally to the more general *PER(A)*, for  $A$  a combinatory algebra, as in Remark 2.3.

A fundamental subcategory of **PER** is **I**, the category whose objects are all pers (on  $\mathbb{N}$ ) but whose only maps are named by the identity function on the natural numbers  $\mathbb{N}$ . Note that there is at most one map in **I** between two pers  $E$  and  $E'$ . Any category with this property can be identified with a partially ordered set (poset), in this case the poset of pers ordered by inclusion. It is misleading, however, to call the maps of **I** *inclusion* maps. They need not be monomorphisms (Note that every per is included in the maximal per; the corresponding map in **I** is the map that collapses an object into  $\mathbf{1}$ . See also Example 2.5).

Even though **I** is a small part of **PER**, it is a representative part.

**2.4. Proposition.** *Every morphism in PER may be decomposed into an isomorphism followed by an I-map followed by an isomorphism.*

**Proof.** Let  $f: E \rightarrow E'$  be a morphism of pers. Consider the following factorization

$$\begin{array}{ccc} E & \longrightarrow & E' \\ \downarrow & & \uparrow \\ D & \longrightarrow & D' \end{array}$$

where

- (i)  $\langle n, k \rangle D \langle n', k' \rangle$  iff  $n E n', k E' f(n)$ , and  $k' E' f(n')$ ,
- (ii)  $\langle n, k \rangle D' \langle n', k' \rangle$  iff  $k E' k'$ .

Note that  $D$  is the “graph” of  $f$ . One easily checks that  $E$  is isomorphic to  $D$ ,  $D \subseteq D'$  and  $D'$  is isomorphic to  $E'$ .

**2.5. Example.** To illustrate Proposition 2.4, consider the pers  $E = \{(0, 0), (1, 1)\}$ ,  $E' = \{(2, 2)\}$ , and let  $f: E \rightarrow E'$  be the constant recursive function  $\lambda x.2$ . Notice  $\text{dom}_E = \{0, 1\}$ ,  $\text{dom}_{E'} = \{2\}$ . Factor  $f$  as above. We obtain  $D = \{((0, 2), (0, 2)), ((1, 2), (1, 2))\}$ . So  $\text{dom}_D = \{(0, 2), (1, 2)\} = \{\langle n, f(n) \rangle \mid n \in \text{dom}_E\}$ , the “graph” of  $f$ . Hence  $E \cong D$ , via the map  $n \mapsto \langle n, f(n) \rangle$ .  $D' = \{\langle n, 2 \rangle, \langle m, 2 \rangle \mid m, n \in \mathbf{N}\}$ . Obviously  $D \subseteq D'$ . Finally,  $D' \cong E'$  by the projection  $\langle n, 2 \rangle \mapsto 2$ . Note that  $\text{dom}_D/D$  has two elements,  $\text{dom}_{D'}/D'$  has one element, so the  $I$ -map morphism  $D \rightarrow D'$  is *not* mono (in  $PER$ ).

**2.6. Definition.** A *realizable* functor  $F: PER \rightarrow PER$  is one which takes  $I$  to  $I$  and for which there exists a mapping  $\Phi$  from the set of partial recursive functions to itself such that for any morphism of pers from  $E$  to  $E'$  named by  $f$ ,  $F(f)$  is named by  $\Phi(f)$ .

Almost any functor which arises in practice is realizable. Realizable functors are closed under products, twisted exponentials (see 1.5), and of course under substitution. Indeed, any functor definable in polymorphic lambda calculus is realizable (see below).

**Remark.** Among the realizable functors are those functors on  $PER$  that are given internally in a critically important model for intuitionistic set theory and higher order logic, the Realizability Universe (or Effective Topos). As first pointed out by Moggi, this Universe contains a non-trivial, complete small cartesian closed category, the *Modest Sets* [36, 8, 28, 29, 57]. (For a discussion of various notions of internal completeness satisfied by Modest Sets, cf. [52].) Viewed externally, Modest Sets are equivalent to  $PER$ . This point of view was instrumental to our initial understanding of the approach described in this paper.

**2.7. Definition.** Let  $F, G: (PER^\circ)^n \times PER^n \rightarrow PER$  be realizable functors. A family  $u = \{u_A: FAA \rightarrow GAA \mid A \in PER^n\}$ , *not* necessarily dinatural, is called a *realizable family* if there is a single partial recursive function  $\varphi$  such that each component  $u_A$  is named by  $\varphi$ .

So a realizable family  $u = \{u_A: FAA \rightarrow GAA\}$  has the property that there is a *single* numerical code  $n$  such that all components  $u_A$  are named by the  $n$ th partial recursive function.

In the following proposition, we refer to the notation introduced before Proposition 1.3.

**2.8. Proposition.** *For any realizable family  $u$ ,  $D_u$  contains  $I$ .*

**Proof.** Immediate.  $\square$

We do not know if realizable families are necessarily dinatural.

We now state the first fundamental theorem of this approach to semantics of polymorphism.

**2.9. Theorem.** *Realizable dinatural transformations compose.*

**Proof.** Using the notation introduced before Proposition 1.3, suppose  $u$  and  $v$  are realizable dinatural. The composition  $u;v$  is of course realizable and therefore by Proposition 2.8,  $D_{u;v}$  contains  $I$ . Proposition 1.4 says that  $D_{u;v}$  contains all isomorphisms. Propositions 2.4 and 1.3 yield the theorem.

**2.10. Corollary.** *For each  $n$ , the realizable functors  $(PER^o)^n \times PER^n \rightarrow PER$  and realizable dinatural transformations between them are a ccc.*

**Proof.** That we have a category is immediate from the theorem. The ccc structure is given by products and twisted exponentials described in Section 1.5.  $\square$

We now relativize the notion of end discussed in Section 1.5 by restricting functors to realizable functors, and dinatural transformations to realizable transformations. Henceforth,  $\int_A GAA$  will denote these *realizable ends*.

The second fundamental theorem of this approach is as follows.

**2.11. Theorem.** *Realizable ends exist.*

**Proof.** The per  $\int_A GAA$  is obtained by first taking the intersection of all pers  $GAA$ , then taking the subper corresponding to the dinaturality condition. Formally, take the per  $E$  that relates  $m$  to  $n$  iff for any  $f:A \rightarrow A'$ ,  $GAA'$  relates  $G(A, f)(m)$  to  $G(f, A')(n)$ . By specializing  $f$  to the identity map, one can easily obtain that the map  $E \rightarrow G(A, A)$  is in  $I$  for every  $A$ , and hence that the induced wedge into  $G$  is realizable, named by the identity function on  $\mathbb{N}$ . A routine calculation shows that  $E$  is a realizable end  $\int_A GAA$ . In fact, the realizable dinatural transformation from  $K_D \rightarrow K_E$  required in Corollary 2.10 is named by the *same* partial recursive function as the given realizable dinatural transformation  $K_D \rightarrow G$ . This property implies that, if there are other variables, then  $\int_A GAA$  takes  $I$  to  $I$ . In that case, furthermore,  $\int_A GAA$  is a realizable functor in these other variables, the mapping on partial recursive functions required for realizability being given by  $\Phi'(g) = \Phi(\text{id}_A, \text{id}_A, g)$ , where  $\Phi$  is assumed by the realizability of  $G$ .  $\square$

**2.12. Remark.** It is a remarkable fact that the realizable end  $\int_A (A \Rightarrow A) \Rightarrow (A \Rightarrow A)$  is the per given by ordinary equality on the natural numbers  $\mathbb{N}$ . This is true even if we substitute the intersection  $\bigcap_A$  instead of the realizable end  $\int_A$ . This should

be contrasted with the situation in SET: the end  $\int_A (A \Rightarrow A) \Rightarrow (A \Rightarrow A)$  is non-existently large, i.e. there are actually a proper class of dinatural transformations from  $( )^{(1)}$  to itself (See Appendix A.5).

**Open problem.** Are realizable families automatically dinatural?

### 3. Interpreting the syntax

Following the viewpoint of Sections 1 and 2, we interpret types  $\sigma$ , whose free type variables are among  $\alpha_1, \alpha_2, \dots, \alpha_k$ , as realizable functors  $|\sigma| : (\mathbf{PER}^\circ)^k \times \mathbf{PER}^k \rightarrow \mathbf{PER}$  by induction.

(1) If  $\sigma = \alpha_i$ ,  $|\sigma|(A, B) = B_i$ , the projection functor onto the  $i$ th covariant argument.

(2) If  $\sigma = C$ , a ground constant interpreted as an object in  $\mathbf{PER}$ , then  $|\sigma|(A, B) = K_C$ , the constant functor with value  $C$ .

(3) If  $\sigma = \tau_1 \times \tau_2$ ,  $|\sigma|(A, B) = (|\tau_1| \times |\tau_2|)(A, B) = |\tau_1|(A, B) \times |\tau_2|(A, B)$ .

(4) If  $\sigma = \tau_1 \Rightarrow \tau_2$ ,  $|\sigma|(A, B) = (|\tau_1| \Rightarrow |\tau_2|)(A, B) = |\tau_1|(B, A) \Rightarrow |\tau_2|(A, B)$ , (twisted exponential).

(5) If  $\sigma = \forall \alpha_i, \tau$ ,  $|\sigma|(A, B) = \int_Q |\tau|(A[i := Q], B[i := Q])$  (realizable end), where  $A[i := Q]$  means change component  $i$  to  $Q$  in vector  $A$ .

**Example.**  $|\forall \alpha(\alpha \times (\alpha \Rightarrow \beta))| : (\mathbf{PER}^\circ)^2 \times \mathbf{PER}^2 \rightarrow \mathbf{PER}$  is given by

$$\begin{aligned} & |\forall \alpha(\alpha \times (\alpha \Rightarrow \beta))|(A_1 A_2, B_1 B_2) \\ &= \int_Q |(\alpha \times (\alpha \Rightarrow \beta))|(A_1 A_2[1 := Q], B_1 B_2[1 := Q]) \\ &= \int_Q |(\alpha \times (\alpha \Rightarrow \beta))|(Q A_2, Q B_2) = \int_Q Q \times (Q \Rightarrow B_2). \end{aligned}$$

We now show how to interpret terms in the dinatural calculus over  $\mathbf{PER}$ . Given a derivable typing judgement in second order polymorphic lambda calculus (e.g. [41, 53])

$$x_1 : \sigma_1, x_2 : \sigma_2, \dots, x_n : \sigma_n \vdash t : \tau,$$

in which all the free type variables are among  $\alpha = \alpha_1, \dots, \alpha_k$ , we define a numerical code  $e_t$  of an  $n$ -ary partial recursive function  $e_t$  by *erasing types from  $t$  and then interpreting the obtained untyped lambda term as a numerical code of a partial recursive function in the usual way* (see Appendix A.1).

Note that  $e_t$  depends only on  $n$  and the untyped lambda term obtained by type erasure, not on any type information.

**Soundness Theorem** (Realizable dinatural transformations). *Consider a derivable typing judgement*

$$x_1 : \sigma_1, x_2 : \sigma_2, \dots, x_n : \sigma_n \vdash t : \tau$$

*in which all free type variables are among  $\alpha = \alpha_1, \dots, \alpha_k$ . Then  $e_t$  names a realizable dinatural transformation  $|(\sigma_1 \times \dots \times \sigma_n)| \rightarrow |\tau| : (\mathbf{PER}^\circ)^k \times \mathbf{PER}^k \rightarrow \mathbf{PER}$ , i.e. a realizable dinatural family  $|(\sigma_1 \times \dots \times \sigma_n)|(A, A) \rightarrow |\tau|(A, A)$ , for every  $A \in \mathbf{PER}^k$ . Furthermore, if  $t_1$  and  $t_2$  are second-order  $\beta$ - $\eta$ -convertible, then  $e_{t_1}$  and  $e_{t_2}$  name the same family.*

**Proof.** By routine induction on the derivation of typing judgements, one verifies that the number  $e_t$  names the dinatural structure referred to in Corollary 2.10 and Theorem 2.11.  $\square$

**3.1. Remark.** Dinaturality imposes certain equations, which, by Soundness, are valid in this interpretation. For example,  $\forall \alpha (\alpha \Rightarrow \alpha)$ , in the realizable end interpretation, must be 1. This imposes the equation  $z = \Lambda \alpha (\lambda_{x:\alpha}. x)$ , for any  $z$  of type  $\forall \alpha (\alpha \Rightarrow \alpha)$ .

**3.2. Remark.** The above interpretation can be considered as part of a general functorial/dinatural interpretation of types and terms, as in Section 1, specialized to  $\mathbf{PER}$ . See also the recent work of Freyd [18].

## 4. Reynolds parametricity in pers

### 4.1. Saturated relations on pers

The category  $\mathbf{PER}$  is rich enough to allow itself a calculus of relations (see [8]). A relation from one per to another is a subobject of their product. For present purposes we will not be interested in all such, but only in those that satisfy the technical condition of being regular. (For the specialists in category theory, happily the two common senses of regular subobject coincide here: a subobject of a per appears as an equalizer of two maps between pers iff it is closed in the double negation topology on the Realizability Universe, i.e. the Effective Topos). For our purposes a regular subobject of a per  $A$  will be defined as one that can be obtained by restricting  $A$  to a ‘‘saturated’’ subset of  $\text{dom}_A$  (where ‘‘saturated’’ means that it is a union of  $A$ -equivalence classes). A regular relation from  $A$  to  $A'$ , therefore, is one that is obtained from a saturated subset of  $\text{dom}_{A \times A'}$ . Such a subset may be viewed as an ordinary relation  $S$  from  $\text{dom}_A$  to  $\text{dom}_{A'}$  subject to the saturation condition which may now be rewritten as:

$$S = A; S; A' \tag{\S}$$

or equivalently,

$$j A n, n S n', n' A' j' \text{ imply } j S j'.$$

**4.2. Example.** Consider a morphism of pers  $f: A \rightarrow A'$ . The graph of  $f$  is a subobject of  $A \times A'$  which appears as the image of the map  $\langle 1, f \rangle: A \rightarrow A \times A'$ . This is a regular subobject, but for no particular name of  $f$  need the set of ordered pairs  $\langle a, f(a) \rangle$ ,  $a \in \text{dom}_A$ , be saturated. The *saturation* of this set is independent of the choice of name of  $f$ . The corresponding saturated relation  $R_f$  from  $\text{dom}_A$  to  $\text{dom}_{A'}$  is equal to the composition of  $f$  with  $A'$ . In other words,  $a R_f a'$  iff  $f(a) A' a'$ .

**4.3. Example.** As a special case of Example 4.2, the identity relation on  $A$  is the graph of the identity map. The corresponding saturated relation is  $A$  itself (condition (§) may be reread as saying that  $S$  is a relation that makes what we have just constructed as the identity morphism behave like the identity morphism).

We shall be working with the Reynolds parametricity condition [48] in the context of saturated relations on pers. One could, of course, simply interpret the Reynolds parametricity condition in the intuitionistic logic of the Realizability Universe, where types are interpreted as modest sets. Under such an interpretation, parametric elements must be invariant under *all* binary relations on modest sets in the Realizability Universe. However, we shall consider an *a priori* weaker invariance requirement, an invariance under saturated (i.e. double negation closed) relations on modest sets. This approach still suffices for proving the theorem in Section 4.9 and will perhaps be more accessible to the reader. Furthermore, considering *only* double negation closed relations is in the spirit of the definition of modest sets themselves as subquotients of the set of natural numbers by double negation closed equivalence relations.

Throughout this section, we shall use the notation  $R: A \leftrightarrow A'$  for a saturated relation from  $\text{dom}_A$  to  $\text{dom}_{A'}$ .

In order to state the Reynolds parametricity conditions in *PER*, we must extend the type-forming operations to saturated relations as follows.

Let  $A, A', B, B'$  be pers and let  $R: A \leftrightarrow A', S: B \leftrightarrow B'$ . Product and exponentiation of pers  $A \times B$  and  $A \Rightarrow B$  are defined as usual (see the proof of Proposition 2.2). Let  $a \in \text{dom}_A, b \in \text{dom}_B, a' \in \text{dom}_{A'}, b' \in \text{dom}_{B'}$ . Define the following relations.

$$R \times S: (A \times B) \leftrightarrow (A' \times B'), \text{ where } \langle a, b \rangle (R \times S) \langle a', b' \rangle \text{ iff } a R a' \text{ and } b S b'.$$

$R \Rightarrow S: (A \Rightarrow B) \leftrightarrow (A' \Rightarrow B')$ , where  $e (R \Rightarrow S) e'$  iff  $(a R a')$  implies  $(ea S e'a')$  for any  $a, a'$  as above.

Suppose, given a type expression  $\tau(\alpha, \beta)$  and pers  $A, B_1, \dots, B_n$ , we know the meaning of  $\tau(A, \mathbf{B})$ . Then let  $\forall \alpha \tau(\alpha, \mathbf{B}) = \bigcap_A \tau(A, \mathbf{B})$ , the intersection over all pers  $A$ . Given a vector of saturated relations  $\mathbf{S}$ , where  $S_i: B_i \leftrightarrow B'_i$ , suppose we know the meaning of  $\tau(R, \mathbf{S}): \tau(A, \mathbf{B}) \leftrightarrow \tau(A', \mathbf{B}')$ . Define

$\forall \alpha. \tau(\alpha, \mathbf{S}) : \forall \alpha. \tau(\alpha, \mathbf{B}) \leftrightarrow \forall \alpha. \tau(\alpha, \mathbf{B}')$  as the intersection  $\bigcap_R \tau(R, \mathbf{S})$ , over all pers  $A, A'$  and all saturated relations  $R$  from  $A$  to  $A'$ . We will want

(i) whenever  $R$  is a saturated relation from  $A$  to  $A'$ , then for any polymorphic type  $\tau$   $\tau(R)$  is a saturated relation from  $\tau(A)$  to  $\tau(A')$ , and the

(ii) Identity Extension Lemma: if  $R$  is the identity on  $A$ , then  $\tau(R)$  is the identity on  $\tau(A)$ .

The second requirement forces us to redefine the per interpretation of  $\forall \alpha. \tau$ , i.e. to trim down the intersection  $\bigcap_A \tau(A)$  to only those elements invariant with respect to all saturated relations.

Officially,  $\tau(A)$  and  $\tau(\mathbf{R})$  are defined, and (i) proved, simultaneously by induction on the complexity of  $\tau$ . The basic clauses for product and function type are the same as before. The formula for universal type abstraction is:  $n \forall \alpha \tau(\alpha, \mathbf{B}) k$  iff for all pers  $A, A'$  and all saturated relations  $R$  from  $A$  to  $A'$ ,

$$n \tau(R, \text{id}_B) n, \quad k \tau(R, \text{id}_B) k, \quad n \tau(A, \mathbf{B}) k, \quad n \tau(A', \mathbf{B}) k.$$

That is, given  $n, k$  in the intersection of the domains of the  $\tau(A, \mathbf{B})$ s, all  $A$ , if  $n$  is to be related to  $k$  by  $\forall \alpha \tau(\alpha, \mathbf{B})$ , it must at least be the case that  $n \tau(A, \mathbf{B}) k$  and  $n \tau(A', \mathbf{B}) k$  for all  $A$  and  $A'$ . But we want more; namely, consider any saturated relation  $R : A \rightarrow A'$  and the induced regular relation between pers,  $\bar{\tau}(R, \text{id}_B)$  from  $\tau(A, \mathbf{B})$  to  $\tau(A', \mathbf{B})$ , corresponding to the saturated relation  $\tau(R, \text{id}_B) : \tau(A, \mathbf{B}) \rightarrow \tau(A', \mathbf{B})$  by Section 4.1. We have at least  $\langle n, n \rangle (\tau(A, \mathbf{B}) \times \tau(A', \mathbf{B})) \langle k, k \rangle$ , but we stipulate that  $\langle n, n \rangle \bar{\tau}(R, \text{id}_B) \langle k, k \rangle$ . Now recall that according to Section 4.1, since  $n$  is in the domain of every  $\tau(A, \mathbf{B})$ , all  $A$ , this means  $n \tau(R, \text{id}_B) n$ ,  $n \tau(A, \mathbf{B}) k$ , and  $n \tau(A', \mathbf{B}) k$ , and hence  $k \tau(R, \text{id}_B) k$  as well, assuming (i) as the induction hypothesis. For the record, the relation  $\forall \alpha. \tau(\alpha, \mathbf{S}) : \forall \alpha. \tau(\alpha, \mathbf{B}) \leftrightarrow \forall \alpha. \tau(\alpha, \mathbf{B}')$  is defined as before by intersection.

The reason for this definition is as follows.

**4.4. Proposition.** *Let  $\tau$  be a second-order polymorphic type whose free type variables are  $\alpha_1, \alpha_2, \dots, \alpha_n$ . Let  $\mathbf{R}$  be a vector of saturated relations, where  $R_i : A_i \rightarrow A'_i$ ,  $i = 1, 2, \dots, n$ . Then  $\tau(\mathbf{R})$  is a saturated relation from  $\tau(A)$  to  $\tau(A')$ .*

**Proof.** Immediate for products and function types. By construction for universal type abstraction (see above).  $\square$

**4.5. Example.** *It is built into our construction that Church numerals are the only elements of  $\text{dom}_{\forall \alpha ((\alpha \Rightarrow \alpha) \Rightarrow (\alpha \Rightarrow \alpha))}$ . Indeed, any element  $k$  must have the property that for any pers  $A, A'$ , and any saturated relation  $R : A \rightarrow A'$ ,  $k ((R \Rightarrow R) \Rightarrow (R \Rightarrow R)) k$ . Therefore,  $k$  is a code of a partial recursive function  $\varphi$  such that if  $e (R \Rightarrow R) e'$  then  $\varphi(e)$  and  $\varphi(e')$  are defined and  $\varphi(e) (R \Rightarrow R) \varphi(e')$ . Recall that the saturated relation  $R \Rightarrow R : (A \Rightarrow A) \rightarrow (A' \Rightarrow A')$  is defined by: if  $e \in A \Rightarrow A$ , and  $e' \in A' \Rightarrow A'$ ,  $e (R \Rightarrow R) e'$  iff whenever  $a R a'$ , it must be the case that  $ea R e'a'$ . Therefore,*

$$\begin{aligned} &\text{if (for all } a, a', \text{ whenever } a R a' \text{ then } ea R e'a') \\ &\text{then (for all } a, a' \text{ if } a R a' \text{ then } \varphi(e)(a) R \varphi(e')(a')). \end{aligned} \quad (\dagger)$$



Now let  $R$  be the saturated relation  $R_f$  corresponding to the graph of a map  $f: A \rightarrow A'$ , i.e.  $a R_f a'$  iff  $f(a) A' a'$  (cf. Example 4.2). Thus condition ( $\dagger$ ) now says:

if (for all  $a, a'$ , whenever  $f(a) A' a'$ , then  $f(ea) A' e'a'$ ,  
 i.e.  $f(ea) A' e'(fa)$ , for all  $a$ )  
 then (for all  $a, a'$ , whenever  $f(a) A' a'$ , then  $f(\varphi(e)(a)) A' \varphi(e')(a')$ , i.e.  
 $f(\varphi(e)(a)) A' \varphi(e')(f(a))$ , for all  $a$ ).

That is, in diagrammatic notation:

$$\text{if } \begin{array}{ccc} A & \xrightarrow{e} & A \\ f \downarrow & & \downarrow f \\ A' & \xrightarrow{e'} & A' \end{array} \quad \text{then} \quad \begin{array}{ccc} A & \xrightarrow{\varphi(e)} & A \\ f \downarrow & & \downarrow f \\ A' & \xrightarrow{\varphi(e')} & A' \end{array} \quad (\dagger\dagger)$$

It now easily follows that  $\varphi$  is the Church numeral  $n$ , where  $n$  may be computed by applying  $\varphi$  to a code  $e$  of the successor function on the natural numbers, and then computing  $n = \varphi(e)(0)$ . Indeed, we show that for any per  $A'$ , and any map  $A' \rightarrow A'$  named by a code  $e'$ , and any  $a' \in \text{dom}_{A'}$ ,  $A'$  relates  $\varphi(e')(a')$  to the  $n$ -fold application  $e'(e'(e' \dots (e'a') \dots))$ . In the condition ( $\dagger\dagger$ ), let the per  $A$  be the ordinary equality considered as an equivalence relation on the natural numbers  $\mathbb{N}$  and let  $e$  be a code of the successor function. Let  $a' \in \text{dom}_{A'}$  and let  $f: A \rightarrow A'$  be constructed by primitive recursion:  $f(0) = a'$  and  $f(i+1) = e'(f(i))$ , so that  $f$  satisfies the assumption in the condition ( $\dagger\dagger$ ). Therefore by ( $\dagger\dagger$ ),  $A'$  relates  $\varphi(e')(a')$  to  $f(\varphi(e)(0))$ , i.e. to  $f(n)$ . But the definition of  $f(b)$  is exactly the required  $n$ -fold application  $e'(e'(e' \dots (e'a') \dots))$ .

Recalling Example 4.3, we now prove the following.

**4.6. Proposition (Identity Extension Lemma).** *Let  $\tau(\alpha_1, \alpha_2, \dots, \alpha_n)$  be a second-order polymorphic type with free type variables  $\alpha_1, \alpha_2, \dots, \alpha_n$  and let  $A_1, \dots, A_n$  be pers. Then  $\tau(\text{id}_{A_1}, \text{id}_{A_2}, \dots, \text{id}_{A_n}) = \text{id}_{\tau(A_1, A_2, \dots, A_n)}$ , as saturated relation:  $\tau(A_1, \dots, A_n) \leftrightarrow \tau(A_1, \dots, A_n)$ .*

**Proof.** Let us do the *abstraction* case first. Let  $\mathbf{B} = B_1, \dots, B_n$  be a list of pers. We wish to show the relation  $\forall \alpha \tau(\alpha, \text{id}_{\mathbf{B}}) = \text{id}_{\forall \alpha \tau(\alpha, \mathbf{B})}$ , as relations  $\forall \alpha \tau(\alpha, \mathbf{B}) \leftrightarrow \forall \alpha \tau(\alpha, \mathbf{B})$ . Let  $n, k \in \text{dom}_{\forall \alpha \tau(\alpha, \mathbf{B})}$ .  $n \forall \alpha \tau(\alpha, \mathbf{B}) k$  iff for every pair of pers  $A, A'$  and saturated  $R: A \leftrightarrow A'$ ,  $n \tau(R, \text{id}_{\mathbf{B}}) n$ ,  $k \tau(R, \text{id}_{\mathbf{B}}) k$ ,  $n \tau(A, \mathbf{B}) k$ ,  $n \tau(A', \mathbf{B}) k$ , whereas  $n \forall \tau(\alpha, \text{id}_{\mathbf{B}}) k$  iff by definition, for every pair of pers  $A, A'$  and saturated relation  $R: A \leftrightarrow A'$ ,  $n \tau(R, \text{id}_{\mathbf{B}}) k$ .  $\forall \alpha \tau(\alpha, \text{id}_{\mathbf{B}})$  is included in  $\text{id}_{\forall \alpha \tau(\alpha, \mathbf{B})}$ , by 4.3, since if  $n \forall \alpha \tau(\alpha, \text{id}_{\mathbf{B}}) k$  then for any saturated  $R: A \leftrightarrow A'$ ,  $n \tau(R, \text{id}_{\mathbf{B}}) k$ . Letting  $R \equiv \text{id}_A$  and the induction hypothesis, gives  $n \tau(A, \mathbf{B}) k$ . Similarly, using  $\text{id}_{A'}$ ,  $n \tau(A', \mathbf{B}) k$ . By saturation,  $k \tau(R, \text{id}_{\mathbf{B}}) k$  and ditto for  $n$ . For the converse inclusion use the inductive hypothesis, setting  $R \equiv \text{id}_A$ ,  $R \equiv \text{id}_{A'}$ .

*Function-type:* for variety, we illustrate this case with an example (which, however, contains all the essential difficulties). Consider  $\text{id}_A \Rightarrow \text{id}_B : (A \Rightarrow B) \leftrightarrow (A \Rightarrow B)$ . Then if  $e, e' \in A \Rightarrow B$ ,  $e (\text{id}_A \Rightarrow \text{id}_B) e'$  iff for all  $a, a'$  in  $\text{dom}_A$ , if  $a \text{id}_A a'$  then  $ea \downarrow$ ,  $ea' \downarrow$ , and  $ea \text{id}_B e'a'$ . That is, by 4.2 if  $a A a'$  then  $ea B e'a'$  (up to definedness of  $ea, e'a'$ ). In particular, setting  $a \equiv a'$ , we see: for all  $a$  such that  $ea \downarrow$ ,  $e'a' \downarrow$ ,  $ea = e'a$  (that is,  $e$  and  $e'$  are extensionally equal). Now suppose  $a A a'$ . Then  $e$ , being a morphism, satisfies  $ea A ea'$  (ignoring definedness). Since  $ea = e'a'$ , we have  $ea A e'a'$ . But this is precisely the meaning of equality of morphisms  $e, e' : A \rightarrow B$ . So  $\text{id}_A \Rightarrow \text{id}_B$  equals  $\text{id}_{A \Rightarrow B}$ .

*Product types:* easy.  $\square$

**Remark.** We denote the identity relation on a per  $A$  by  $A$  itself. So by the lemma above, we may write  $\tau(A)$  for  $\tau(\text{id}_{A_1}, \text{id}_{A_2}, \dots, \text{id}_{A_n})$ .

#### 4.7. Every polymorphic term is invariant under saturated relations in *PER*

Given a derivable typing judgement in second-order polymorphic lambda calculus (e.g. [41, 53])

$$x_1:\sigma_1, x_2:\sigma_2, \dots, x_n:\sigma_n \vdash t:\tau$$

in which all the free type variables are among  $\alpha = \alpha_1, \dots, \alpha_k$ , we define a numerical code  $e_t$  of an  $n$ -ary partial recursive function  $e_t$  by *erasing types from  $t$  and then interpreting the obtained untyped lambda term as a numerical code of a partial recursive function in the usual way* (see Appendix A.1).

Again note that  $e_t$  depends only on  $n$  and the untyped lambda term obtained by type erasure, not on any type information.

In Section 3 we showed that  $e_t$  is a realizable dinatural transformation from  $\sigma_1 \times \dots \times \sigma_n$  to  $\tau$  (see Soundness Theorem). Here we wish to show that  $e_t$  satisfies our version of the Reynolds parametricity condition in *PER*.

**Soundness Theorem.** *Consider a derivable typing judgement*

$$x_1:\sigma_1, x_2:\sigma_2, \dots, x_n:\sigma_n \vdash t:\tau$$

*in which all free type variables are among  $\alpha = \alpha_1, \dots, \alpha_k$ . Let  $A_1, A'_1, \dots, A_k, A'_k$  be arbitrary pers and let  $R_i : A_i \leftrightarrow A'_i$ ,  $i = 1, \dots, k$  be arbitrary saturated relations. Then  $e_t$  names per maps*

$$(\sigma_1 \times \dots \times \sigma_n)(A) \rightarrow \tau(A) \quad \text{and} \quad (\sigma_1 \times \dots \times \sigma_n)(A') \rightarrow \tau(A')$$

*such that  $m (\sigma_1 \times \dots \times \sigma_n)(R) m'$  implies  $e_t(m) \tau(R) e_t(m')$ .*

*Furthermore, if  $t_1$  and  $t_2$  are  $\beta$ - $\eta$ -convertible, then  $e_{t_1}$  and  $e_{t_2}$  name the same map of pers  $(\sigma_1 \times \dots \times \sigma_n)(A) \rightarrow \tau(A)$  for every  $A$ .*

**Note.** In diagrammatic language in *PER*

$$\begin{array}{ccc}
 (\sigma_1 \times \cdots \times \sigma_n)(A) & \xrightarrow{e_t} & \tau(A) \\
 \nearrow & & \nearrow \\
 (\sigma_1 \times \cdots \times \sigma_n)(R) & \dashrightarrow^{\exists} & \tau(R) \\
 \searrow & & \searrow \\
 (\sigma_1 \times \cdots \times \sigma_n)(A') & \xrightarrow{e_t} & \tau(A')
 \end{array}$$

**Note.** the factorizing map of pers  $(\sigma_1 \times \cdots \times \sigma_n)(R) \rightarrow \tau(R)$  is not necessarily named by  $e_t$ .

Before proving the theorem, we recall Example 4.5 and the notation from Example 4.2. Therefore a typing judgement  $x: \alpha \Rightarrow \alpha \vdash t: \alpha \Rightarrow \alpha$  is necessarily interpreted as a Church numeral.

**Proof of Soundness Theorem.** By induction on the derivation of typing judgements. We give some representative cases. The reader is assumed to have looked at Appendix A.1.

$\Rightarrow$ -introduction: Suppose the statement holds for

$$x_1: \sigma_1, x_2: \sigma_2, \dots, x_n: \sigma_n, x: \sigma \vdash t: \tau$$

We want to prove the statement for

$$x_1: \sigma_1, x_2: \sigma_2, \dots, x_n: \sigma_n \vdash \lambda x: \sigma. t: \sigma \Rightarrow \tau.$$

Suppose  $i_j \sigma_j(R) i'_j$ , where  $j \in \{1, \dots, n\}$ . Suppose  $l \sigma(R) l'$ . By the induction hypothesis,  $e_t(i_1, \dots, i_n, l) \tau(R) e_t(i'_1, \dots, i'_n, l')$ . We want to show that

$$S_1^n(e_t, i_1, \dots, i_n) (\sigma \Rightarrow \tau)(R) S_1^n(e_t, i'_1, \dots, i'_n).$$

First notice that  $S_1^n(e_t, i_1, \dots, i_n)$  is in the domain of  $(\sigma \Rightarrow \tau)(A)$ . Indeed, suppose  $l_1 \sigma(A) l_2$ . Then by the induction hypothesis,  $\overline{e_t}(i_1, \dots, i_n, l_1)$  and  $e_t(i_1, \dots, i_n, l_2)$  are both defined and are related in  $\tau(A)$ . In other words,  $S_1^n(e_t, i_1, \dots, i_n)(l_1)$  and  $S_1^n(e_t, i_1, \dots, i_n)(l_2)$  are defined and related in  $\tau(A)$ . Similar considerations apply to  $(\sigma \Rightarrow \tau)(A')$ . We have to show that  $S_1^n(e_t, i_1, \dots, i_n)(l) \tau(R) S_1^n(e_t, i'_1, \dots, i'_n)(l')$ . This follows by the induction hypothesis because, by the *S-m-n* theorem,  $S_1^n(e_t, i_1, \dots, i_n)(l) \cong e_t(i_1, \dots, i_n, l)$  and  $S_1^n(e_t, i'_1, \dots, i'_n)(l') \cong e_t(i'_1, \dots, i'_n, l')$ .

$\forall$ -introduction: Suppose the statement holds for

$$x_1: \sigma_1, x_2: \sigma_2, \dots, x_n: \sigma_n \vdash t: \tau$$

where type variable  $\alpha$  is not free in  $\sigma_1, \sigma_2, \dots, \sigma_n$ . We want to show that the statement holds for

$$x_1:\sigma_1, x_2:\sigma_2, \dots, x_n:\sigma_n \vdash \Lambda\alpha.t:\forall\alpha.\tau.$$

We write  $\tau \equiv \tau(\alpha_i, \alpha)$ . First note that  $e_{\lambda\alpha.t} = e_t$  because  $\Lambda\alpha.t$  and  $t$  have the same type erasures. By the induction hypothesis, for all saturated relations between pers,  $S: B \leftrightarrow B'$

$$\begin{array}{ccc} (\sigma_1 \times \dots \times \sigma_n)(A) & \xrightarrow{e_t} & \tau(A, B) \\ \nearrow & & \nearrow \\ (\sigma_1 \times \dots \times \sigma_n)(R) & \xrightarrow{\exists} & \tau(R, S) \\ \searrow & & \searrow \\ (\sigma_1 \times \dots \times \sigma_n)(A') & \xrightarrow{e_t} & \tau(A', B') \end{array}$$

In particular, specialize to  $A' = A$ ,  $R = \text{id}_A$ . By the Identity Extension Lemma (used on the left-hand side), we have

$$\begin{array}{ccc} & & \tau(A, B) \\ & \nearrow & \\ (\sigma_1 \times \dots \times \sigma_n)(A) & \xrightarrow{\exists} & \tau(A, S) \\ & \searrow & \\ & & \tau(A, B') \end{array}$$

Hence

$$e_t: (\sigma_1 \times \dots \times \sigma_n)(A) \rightarrow \forall\alpha\tau(A, \alpha)$$

and similarly for  $A'$ . The rest is left to the reader.

**$\forall$ -elimination:** Suppose the statement holds for

$$x_1:\sigma_1, x_2:\sigma_2, \dots, x_n:\sigma_n \vdash t:\forall\alpha.\tau.$$

We want to show that it also holds for

$$x_1:\sigma_1, x_2:\sigma_2, \dots, x_n:\sigma_n \vdash t[\beta]:\tau[\alpha := \beta].$$

As above, note that  $e_{t[\beta]} = e_t$  because  $t[\beta]$  and  $t$  have the same type erasures. Argue as in the previous case.  $\square$

We refer to the interpretation just described as the *parametric per interpretation*.

**Open Problem.** We do not know the precise relationship between Reynolds' parametricity and realizable dinaturality. The reader will have observed that the latter is defined by purely semantic means. We do not know a purely semantic definition of the Reynolds parametricity condition, i.e. without recourse to induction on second order polymorphic types.

Now recall the notion of realizable functor from Definition 2.6. Also recall the notion of *positive* (or *covariant*) occurrence of a variable in a type (see [61, Section 1.10.5, p. 86]).

**4.8. Proposition.** Let  $\tau(\alpha, \beta)$  be a second-order polymorphic type whose free type variables are  $\alpha, \beta$ . Suppose  $\alpha$  occurs only positively in  $\tau$ . Then:

(1) For every vector of pers  $\mathbf{B}$ ,  $\tau(\_, \mathbf{B})$  is a realizable functor, whose action on maps,  $\tau(f, \mathbf{B}) = \tau(R_f, \mathbf{B})$ , is independent of  $\mathbf{B}$ .

(2) Furthermore, for any map of pers  $f: A \rightarrow A'$  and for any vector of saturated relations  $\mathbf{S}: \mathbf{B} \leftrightarrow \mathbf{B}'$

$$n \tau(A, \mathbf{S}) n' \text{ implies } \tau(f, \mathbf{B})(n) \tau(A', \mathbf{S}) \tau(f, \mathbf{B})(n').$$

**Remark.** In diagrammatic language in *PER*

$$\begin{array}{ccc}
 \tau(A, \mathbf{B}) & \xrightarrow{\tau(f, \mathbf{B})} & \tau(A', \mathbf{B}) \\
 \nearrow & & \nearrow \\
 \tau(A, \mathbf{S}) & \overset{\exists}{\dashrightarrow} & \tau(A', \mathbf{S}) \\
 \searrow & & \searrow \\
 \tau(A, \mathbf{B}') & \xrightarrow{\tau(f, \mathbf{B}')} & \tau(A', \mathbf{B}')
 \end{array}$$

**Proof.** Similar to the Soundness Theorem, by induction on the complexity of  $\tau$ . We sketch (1) for the basic case of function types. Given  $\sigma(R_f, \mathbf{B}): \sigma(A, \mathbf{B}) \leftrightarrow \sigma(A', \mathbf{B})$  and  $\tau(R_f, \mathbf{B}): \tau(A, \mathbf{B}) \leftrightarrow \tau(A', \mathbf{B})$ , we want to find the action of  $(\sigma \Rightarrow \tau)(f, \mathbf{B})$ , i.e.  $(\sigma \Rightarrow \tau)(R_f, \mathbf{B}): \sigma(A, \mathbf{B}) \Rightarrow \tau(A, \mathbf{B}) \leftrightarrow \sigma(A', \mathbf{B}) \Rightarrow \tau(A', \mathbf{B})$ . The idea is illustrated by the following square

$$\begin{array}{ccc}
 \sigma(A, \mathbf{B}) & \xrightarrow{e} & \tau(A, \mathbf{B}) \\
 \sigma(R_f, \mathbf{B}) \uparrow & & \downarrow \tau(R_f, \mathbf{B}) \\
 \sigma(A', \mathbf{B}) & \xrightarrow{e'} & \tau(A', \mathbf{B})
 \end{array}$$

So define  $e(\sigma(R_f, \mathbf{B}) \Rightarrow \tau(R_f, \mathbf{B})) e'$  iff  $\sigma(R_f, \mathbf{B}); e; \tau(R_f, \mathbf{B}) = e'$ . This formula ( $= \lambda e. \sigma(R_f, \mathbf{B}); e; \tau(R_f, \mathbf{B})$ ) yields the desired action on maps, by the inductive

assumption. The  $\forall$  case follows similarly, along the lines of the Soundness Theorem above.  $\square$

#### 4.9. $\forall\alpha((T\alpha \Rightarrow \alpha) \Rightarrow \alpha)$ is the initial $T$ -algebra

It has been known for some years [4, 48] that for any multisorted finitary algebraic signature, the elements of the initial algebra can be represented as closed normal terms for an appropriate polymorphic type. More generally, let  $T(\alpha)$  be a polymorphic type in which  $\alpha$  occurs only positively. Without imposing further parametricity conditions on universal type abstraction, in general it can only be shown that  $\forall\alpha((T\alpha \Rightarrow \alpha) \Rightarrow \alpha)$  is a *weakly* initial  $T$ -algebra (see below) in any categorical interpretation [51]. Recently, Hyland et al. [30] verified that the standard per interpretation of “algebraic” types (like poly-boole and poly-nat) yields initial algebras in *PER*.

However, we have already seen in Example 4.5 that a systematic parametric per interpretation of universal type abstraction (Sections 4.1–4.5, 4.7) automatically ensures the canonicity of certain universal types. We now wish to explore this further for  $T$ -algebras (see below), where  $T$  is the covariant functor  $PER \rightarrow PER$  induced by a second-order type  $T$ , in the sense of Proposition 4.8. Note that these types include more than just the algebraic ones.

We recall some terminology. If  $T: \mathcal{C} \rightarrow \mathcal{C}$  is a covariant endofunctor, then a  $T$ -algebra is an object  $A$  together with an arrow  $a: TA \rightarrow A$ . Given two such structures  $a: TA \rightarrow A$  and  $b: TB \rightarrow B$ , a  $T$ -algebra *morphism* is an arrow  $h: A \rightarrow B$  such that the following square commutes:

$$\begin{array}{ccc} TA & \xrightarrow{T(h)} & TB \\ a \downarrow & & \downarrow b \\ A & \xrightarrow[h]{B} & B \end{array}$$

$T$ -algebras form a category, and if this category has an *initial object* it is called the *initial  $T$ -algebra*. So, by definition, an initial  $T$ -algebra (if it exists) has a unique morphism to any other  $T$ -algebra. Similarly, *weakly initial* objects have a morphism (not necessarily unique) to any other object.

**Theorem.** *In the parametric  $PER$  interpretation,  $\forall\alpha((T\alpha \Rightarrow \alpha) \Rightarrow \alpha)$  is the initial  $T$ -algebra.*

**Proof.** The interpretation of  $\forall\alpha((T\alpha \Rightarrow \alpha) \Rightarrow \alpha)$  is  $\bigcap_R ((TR \Rightarrow R) \Rightarrow R)$  over all pers  $A, A'$  and all saturated relations  $R: A \leftrightarrow A'$ . We claim this is the initial  $T$ -algebra. We argue similarly to Example 4.5. Any element  $k \in \text{dom}_{\forall\alpha((T\alpha \Rightarrow \alpha) \Rightarrow \alpha)}$  must have the property: for any pers  $A, A'$ , any saturated  $R: A \leftrightarrow A'$ , and any  $a \in T(A) \Rightarrow A$ ,  $a' \in T(A') \Rightarrow A'$ :

if for all  $m \in T(A)$ ,  $m' \in T(A')$  ( $m T(R) m'$  implies  $a m R a' m'$ )  
then  $ke R ke'$ .

Letting  $R \equiv R_f$ , for any per map  $f: A \rightarrow A'$ , we obtain the analog of (††)

$$\text{If } \begin{array}{ccc} T(A) & \xrightarrow{T(f)} & T(A') \\ a \downarrow & & a' \downarrow \\ A & \xrightarrow{f} & A' \end{array} \text{ then } f(ka) A' ka'. \quad (\dagger\dagger\dagger)$$

Therefore, given a morphism of  $T$ -algebras  $f: A \rightarrow A'$ ,  $f(ka) A' ka'$ . Thus  $k$  is the index of a partial recursive function such that for each  $A$ , we have  $k \in (TA \Rightarrow A) \Rightarrow A$ . Hence for each  $T$ -algebra  $a \in TA \rightarrow A$ ,  $ka \in A$ . Let  $G = \forall \alpha ((T\alpha \Rightarrow \alpha) \Rightarrow \alpha)$ . Denote the interpretation of  $G$  by the same letter. Pick the  $k$  that arises, by the Soundness Theorem, from  $e_{t_\alpha}$ , where  $t_\alpha$  is the second-order evaluation  $G \rightarrow ((T\alpha \Rightarrow \alpha) \Rightarrow \alpha)$ . The choices are made to guarantee that (the interpretation of)  $G$  itself is a  $T$ -algebra, whose canonical action  $TG \rightarrow G$  is given by a polymorphic lambda term [51, Proposition 3], such that for any  $T$ -algebra  $a: TA \rightarrow A$ ,  $ka$  induces a morphism of  $T$ -algebras  $G \rightarrow A$ .

We find ourselves in the following situation: the category of  $T$ -algebras over  $PER$  has an object  $G$  with an assigned map to each object. Moreover, it has the property that an assigned map followed by a  $T$ -algebra morphism is (equal to) an assigned map. Now argue as follows: the assigned morphism of  $G$  to itself is necessarily an idempotent (using the properties of the category of  $T$ -algebras just mentioned.). Since  $PER$  has equalizers, it easily follows that idempotents split in the category of  $T$ -algebras over it. Splitting this idempotent of  $G$  yields an initial object. Well-pointedness is used to verify the idempotent in question is necessarily a mono (by verifying it is monomorphic as far as points are concerned). A monomorphic idempotent is trivially the identity.

For a general framework for this kind of argument, see [18].  $\square$

The above theorem has a string of corollaries about the parametric  $PER$  interpretation.

**4.10. Corollary.**  $\forall \alpha (\alpha) \cong 0$ .

**Proof.**  $\forall \alpha (\alpha) \cong \forall \alpha (1 \Rightarrow \alpha) \cong \forall \alpha ((0 \Rightarrow \alpha) \Rightarrow \alpha) \cong 0$ .  $\square$

**4.11. Corollary.**  $\forall \alpha (\alpha \Rightarrow \alpha) \cong 1$ .

**Proof.**  $\forall \alpha (\alpha \Rightarrow \alpha) \cong \forall \alpha ((1 \Rightarrow \alpha) \Rightarrow \alpha) \cong 1$ .  $\square$

**4.12. Corollary.**  $\forall \alpha (\alpha \Rightarrow (\alpha \Rightarrow \alpha)) \cong 1 + 1$ , where  $+$  denotes categorical coproduct.

**Proof.**  $\forall \alpha (\alpha \Rightarrow (\alpha \Rightarrow \alpha)) \cong \forall \alpha ((\alpha \times \alpha) \Rightarrow \alpha) \cong \forall \alpha (((1 + 1) \Rightarrow \alpha) \Rightarrow \alpha) \cong 1 + 1$ .  $\square$

**4.13. Corollary.**  $\forall \alpha((A \Rightarrow \alpha) \Rightarrow ((B \Rightarrow \alpha) \Rightarrow \alpha)) \cong A + B.$

**Proof**

$$\begin{aligned} \forall \alpha((A \Rightarrow \alpha) \Rightarrow ((B \Rightarrow \alpha) \Rightarrow \alpha)) &\cong \forall \alpha(((A \Rightarrow \alpha) \times (B \Rightarrow \alpha)) \Rightarrow \alpha) \\ &\cong \forall \alpha(((A + B) \Rightarrow \alpha) \Rightarrow \alpha) \cong A + B. \quad \square \end{aligned}$$

**4.14. Corollary.**  $\forall \alpha((A \Rightarrow \alpha) \Rightarrow \alpha) \cong A.$

Note that Corollaries 4.10–4.13 are special cases of Corollary 4.14, which is itself a special case of the theorem, with  $T = K_A$ , the constant functor  $A$ . See Appendix A.3 for a description of this phenomenon by means of ends.

**4.15. Corollary.** *Church and Lawvere here agree on the natural numbers. The Church definition is  $\forall \alpha((\alpha \Rightarrow \alpha) \Rightarrow (\alpha \Rightarrow \alpha))$ . The Lawvere definition is the initial  $T$ -algebra for  $TX = 1 + X$ , the categorical coproduct of 1 and  $X$ . In exponential notation:*

$$(X^X)^{X^X} \cong X^{X \times X^X} \cong X^{X^1 \times X^X} \cong X^{X^{1+X}}.$$

*So by the theorem, the initial  $T$ -algebra for the functor  $1 + X$  is isomorphic to  $\forall X.X^{X^{1+X}}$ .*

**4.16. Corollary.** *Associating nested  $\Rightarrow$ 's to the right:*

$$\forall \alpha(\alpha \Rightarrow ((\alpha \Rightarrow \alpha \Rightarrow \alpha) \Rightarrow \alpha)) \cong \forall \alpha(((1 + (\alpha \times \alpha)) \Rightarrow \alpha) \Rightarrow \alpha).$$

**Proof**

$$\begin{aligned} \forall \alpha(\alpha \Rightarrow ((\alpha \Rightarrow \alpha \Rightarrow \alpha) \Rightarrow \alpha)) &\cong \forall \alpha((\alpha \times (\alpha \Rightarrow \alpha \Rightarrow \alpha)) \Rightarrow \alpha) \\ &\cong \forall \alpha((\alpha \times ((\alpha \times \alpha) \Rightarrow \alpha)) \Rightarrow \alpha) \\ &\cong \forall \alpha(((1 + (\alpha \times \alpha)) \Rightarrow \alpha) \Rightarrow \alpha). \quad \square \end{aligned}$$

In the above corollary,  $TX = 1 + (X \times X)$ . The initial  $T$ -algebra consists of binary trees.

**4.17. Corollary.** *For any finite algebraic signature, there exists a functor  $T$  so that  $T$ -algebras are the same as algebras of that signature. As in the special cases of Corollaries 4.10–4.16, the expression  $(TX \Rightarrow X) \Rightarrow X$  may be rewritten as a polymorphic type.*

**4.18. Corollary.**  $\forall \alpha(\alpha \Rightarrow (A \Rightarrow \alpha \Rightarrow \alpha) \Rightarrow \alpha) \cong \forall \alpha(((1 + (A \times \alpha)) \Rightarrow \alpha) \Rightarrow \alpha)$



**Proof**

$$\begin{aligned}
 \forall \alpha (\alpha \Rightarrow (A \Rightarrow \alpha \Rightarrow \alpha) \Rightarrow \alpha) &\cong \forall \alpha ((\alpha \times (A \Rightarrow \alpha \Rightarrow \alpha)) \Rightarrow \alpha) \\
 &\cong \forall \alpha ((\alpha \times ((A \times \alpha) \Rightarrow \alpha)) \Rightarrow \alpha) \\
 &\cong \forall \alpha (((1 + (A \times \alpha)) \Rightarrow \alpha) \Rightarrow \alpha). \quad \square
 \end{aligned}$$

Here  $TX = 1 + (A \times X)$ , for a fixed  $A$ . The initial  $T$ -algebra is  $A$ -list, the algebra of lists with entries from  $A$ , i.e.  $A$ -labelled unary trees.

Similarly, for the functor  $TX = 1 + (A \times X \times X)$ , we obtain the  $A$ -labelled binary trees; for the functor  $TX = B + (A \times X \times X)$ , we obtain binary trees whose binary nodes are labelled by  $A$  and whose leaves are labelled by  $B$ . The polymorphic description of the latter is:

$$\begin{aligned}
 \forall \alpha ((B \Rightarrow \alpha) \Rightarrow (A \Rightarrow \alpha \Rightarrow \alpha \Rightarrow \alpha) \Rightarrow \alpha) &\cong \forall \alpha (((B \Rightarrow \alpha) \times (A \Rightarrow \alpha \Rightarrow \alpha \Rightarrow \alpha)) \Rightarrow \alpha) \\
 &\cong \forall \alpha (((B \Rightarrow \alpha) \times ((A \times \alpha \times \alpha) \Rightarrow \alpha)) \Rightarrow \alpha) \\
 &\cong \forall \alpha (((B + (A \times \alpha \times \alpha)) \Rightarrow \alpha) \Rightarrow \alpha).
 \end{aligned}$$

Furthermore,  $X$ -list is a functor in  $X$ . Given any  $A$ , consider the functor  $TX = A \times (X\text{-list})$ . Its initial algebra consists of arbitrary (rooted) trees with nodes and leaves labelled by  $A$ .

**Appendix**
*A.1. On interpreting terms as partial recursive functions*

Given a derivable typing judgement in second-order polymorphic lambda calculus,

$$x_1:\sigma_1, x_2:\sigma_2, \dots, x_n:\sigma_n \vdash t:\tau$$

we obtain a numerical code  $e_t$  of an  $n$ -ary partial recursive function, by first erasing all the types from  $t$ , so obtaining a term  $a$  ( $\equiv \text{erase}(t)$ ) of untyped  $\lambda$ -calculus, and then interpreting this untyped lambda term to give a number  $e_a$  ( $\equiv e_t$ ) by induction as follows:

If  $a$  is  $x_i$ , then  $e_a$  is (a chosen code of) the  $i$ th projection of  $n$  arguments.

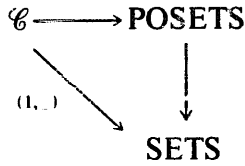
If  $e_a$  is code of a partial recursive function of  $n+1$  arguments, then  $e_{\lambda x.a}$  is defined by using the  $S$ - $m$ - $n$  theorem from recursion theory, as follows. Let  $S_1^n$  be a primitive recursive function of  $n+1$  arguments such that for  $i_1, \dots, i_n$ ,  $S_1^n(e_a, i_1, \dots, i_n)$  is code of a unary partial recursive function such that for all  $b$ ,  $S_1^n(e_a, i_1, \dots, i_n)(b) \equiv e_a(i_1, \dots, i_n, b)$ . Now define  $e_{\lambda x.a}$  as a code (given canonically in a chosen enumeration) of the  $n$ -ary function given as  $S_1^n$  with its first argument held constant at  $e_a$ .

Finally, let  $e_{ab}$  be a code (given canonically in a chosen enumeration) of the function that assigns  $a(j)(b(j))$  to  $j$ .

Note that  $e_t$  depends only on  $n$  and  $\text{erase}(t)$ , not on any type information.

A.2. On least fixed points and dinaturality

Let  $\mathcal{C}$  be a ccc in which, for each  $A$ , the homset  $(1, A)$  is a poset and for which we have a factorization



Furthermore, assume that  $Y \equiv Y_A : A^A \rightarrow A$ , for each  $A$ , takes any argument  $h : A \rightarrow A$  into the *least fixed-point* of  $h$ . We wish to show  $Y$  is dinatural. Let  $f : A \rightarrow B$  and  $g : B \rightarrow A$ . Let  $x = f(Y_A(g \circ f))$ . Claim:  $x$  is a fixed point of  $f \circ g$ . For  $(f \circ g)(x) = f(g \circ f(Y_A(g \circ f))) = f(Y_A(g \circ f)) = x$ . Thus,  $Y_B(f \circ g) \leq f(Y_A(g \circ f))$ , so applying  $g$ ,  $g(Y_B(f \circ g)) \leq g(f(Y_A(g \circ f))) = Y_A(g \circ f)$ . Interchanging the roles of  $f$  and  $g$ , we obtain the inequality in the other direction, viz.  $f(Y_A(g \circ f)) \leq Y_B(f \circ g)$ . Hence  $f(Y_A(g \circ f)) = Y_B(f \circ g)$ , so  $Y$  is dinatural (cf. Example 1.1 (iv)).

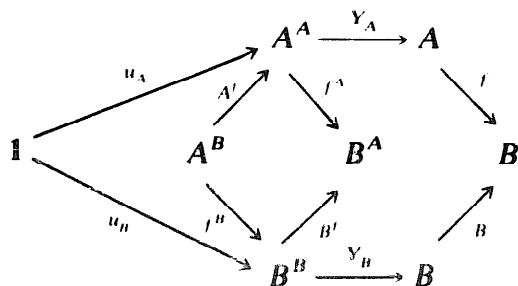
A.3. On the end  $\int_X (X^{X^A})$

Note that in regard to Corollary 4.14, in any well-pointed ccc,  $A \cong \int_X (X^{X^A})$ . (A note on standard categorical terminology: a category is *concrete* if it has an embedding (= faithful functor) into SETS. A category is *well pointed* if it is generated by 1, i.e. the embedding into SETS may be taken to be the external hom-functor  $(1, \_)$ . *PER* is of course well pointed.

A counter-example to the isomorphism above for non-well pointed cccs is given by: let  $G$  be any non-trivial group. In the category  $\text{SET}^G$  ( $G$ -sets) take  $A = 1$ . We obtain that  $\int_X (X^X)$  is  $G$  with the conjugacy action. The forgetful functor  $\text{SET}^G \rightarrow \text{SET}$  is often cited as an example of a functor that preserves everything. It does not, however, preserve  $\int_X (X^X)$ . It would therefore appear not to preserve higher-order types.

A.4. Polymorphic identity followed by  $Y$  yields triviality

Suppose we could form the composite of the dinatural transformations  $u : K_1 \rightarrow (\_)^{(\_)}$  and  $Y : (\_)^{(\_) \rightarrow (\_)}$ . Then the outer hexagon in the following figure must commute, for all objects  $A, B$  and arrows  $A \xrightarrow{f} B$ :



The commutativity of this outer hexagon is expressed, using informal notation, by  $f(Y_A(\lambda x:A.x)) = Y_B(\lambda x:B.x)$ . Now setting  $A \equiv \mathbf{1}$ , we see that every object  $B$  has a *unique* global element  $Y_B(\lambda x:B.x): \mathbf{1} \rightarrow B$ . Thus, given any two objects  $C, D$ , there is a *unique* global element of  $D^C$ ; hence by cartesian-closedness of the underlying category, there is a unique map between any two objects, so the category is trivial.

#### A.5. On dinats from $( )^{(\cdot)}$ to $( )^{(\cdot)}$ .

In SETS, there are a proper class of dinats from  $( )^{(\cdot)}$  to itself. For let  $\kappa$  be any cardinal number and define the family  $\theta_\kappa = \{\theta_\kappa(A): A^A \rightarrow A^A \mid A \in \text{SET}\}$  as follows: let  $h: A \rightarrow A$  and let

$$\theta_\kappa(A)(h) = \begin{cases} h & \text{if } \text{Card}(\text{fix}(h)) = \kappa \\ \text{id}_A & \text{otherwise} \end{cases}$$

where  $\text{fix}(h) = \{a \in A \mid h(a) = a\}$  is the set of *fixed points* of  $h$ .

Dinaturality of the family  $\theta_\kappa: ( )^{(\cdot)} \rightarrow ( )^{(\cdot)}$  in SETS amounts to the following: for any  $g \in B^A$  and  $f \in A^B$ ,  $f \circ (\theta_\kappa(A)(g \circ f)) = ((\theta_\kappa(B)(f \circ g)) \circ f$ . To verify this equation, first observe that  $\text{fix}(g \circ f)$  is in bijective correspondence with  $\text{fix}(f \circ g)$ . Now suppose  $\text{Card}(\text{fix}(g \circ f)) = \text{Card}(\text{fix}(f \circ g)) = \kappa$ . Then the equation becomes  $f \circ (g \circ f) = (f \circ g) \circ f$ , an instance of associativity of composition. On the other hand, if  $\text{Card}(\text{fix}(f \circ g)) \neq \kappa$ , the equation becomes the truism  $f \circ \text{id}_A = \text{id}_B \circ f$ . Since there is a proper class of cardinals, we are done.

Note that by Proposition 1.6, there is a proper class of points

$$\mathbf{1} \rightarrow \int_A (A \Rightarrow A) \Rightarrow (A \Rightarrow A)$$

in SET.

#### A.6. On the functorial calculus

The operations of product and twisted exponential of functors may be described as follows:

**Products:** Given functors  $F, G: (\mathcal{C}^0)^n \times \mathcal{C}^n \rightarrow \mathcal{C}$ , define  $F \times G: (\mathcal{C}^0)^n \times \mathcal{C}^n \rightarrow \mathcal{C}$ , to be the functor

$$(\mathcal{C}^0)^n \times \mathcal{C}^n \xrightarrow{\langle F, G \rangle} \mathcal{C} \times \mathcal{C} \xrightarrow{\times} \mathcal{C}.$$

**Twisted exponentials:** Given functors  $F, G: (\mathcal{C}^0)^n \times \mathcal{C}^n \rightarrow \mathcal{C}$ , define

$$G^F: (\mathcal{C}^0)^n \times \mathcal{C}^n \longrightarrow \mathcal{C} = (\mathcal{C}^0)^n \times \mathcal{C}^n \xrightarrow{\langle F^*, G \rangle} \mathcal{C}^0 \times \mathcal{C} \xrightarrow{(\cdot)^{\circ}} \mathcal{C},$$

where we write  $F^*: (\mathcal{C}^0)^n \times \mathcal{C}^n \rightarrow \mathcal{C}^0$  for the composite

$$(\mathcal{C}^0)^n \times \mathcal{C}^n \cong \mathcal{C}^n \times (\mathcal{C}^0)^n \cong (\mathcal{C}^{00})^n \times (\mathcal{C}^0)^n \cong ((\mathcal{C}^0)^n \times \mathcal{C}^n)^{\circ} \xrightarrow{F^{\circ}} \mathcal{C}^0.$$

**Example.** Covariant projections  $P_i : (\mathcal{C}^\circ)^2 \times \mathcal{C}^2 \rightarrow \mathcal{C}$ ,  $i = 1, 2$ , given by  $P_i(A_1 A_2; B_1 B_2) = B_i$ .

The dinatural transformation:  $\mathbf{app}_{P_1, P_2} : P_2^{P_1} \times P_1 \rightarrow P_2$  is given by  $(\mathbf{app}_{P_1, P_2})_{AA'} : (P_2^{P_1} \times P_1)(AA', AA') \rightarrow P_2(AA', AA')$ , where  $(\mathbf{app}_{P_1, P_2})_{AA'} : A'^A \times A \rightarrow A'$  is application, as in Example 1.1 (iii). Much of the ecc structure of Corollary 2.10 exists for general reasons (see [18]). Additional functorial structure is mentioned in [1].

## Acknowledgment

This paper has benefitted from numerous comments by colleagues at our various conference presentations. We would like to especially thank John Reynolds for his continued interest and insightful comments. Also, thanks to Albert Meyer, John Mitchell, Edmund Robinson, Dana Scott, and the referee for conversations on parametricity and comments on this work. Scott wishes to thank the Department of Mathematics of the University of Pennsylvania for its kind hospitality while this research was being pursued.

## References

- [1] E.S. Bainbridge, P.J. Freyd, A. Scedrov, and P.J. Scott, Functorial polymorphism (Preliminary Report), in: G. Huet, ed., *Logical Foundations of Functional Programming, Proceedings*, University of Texas Programming Institute, Austin, TX (1987).
- [2] H.P. Barendregt, *The Lambda Calculus*, Studies in Logic and the Foundations of Mathematics (North-Holland, Amsterdam, revised ed., 1984).
- [3] J.G.P. Barnes, *Programming in Ada* (Addison-Wesley, Reading, MA, 1981).
- [4] C. Böhm and A. Berarducci, Automatic synthesis of typed  $\lambda$ -programs on term algebras, *Theoret. Comput. Sci.* **39** (1985) 135–154.
- [5] V. Breazu-Tannen and T. Coquand, Extensional models for polymorphism, in: *Proc. TAPSOFT '87-CFLP, Pisa Lecture Notes in Computer Science* **250** (Springer, Berlin, 1987); expanded version in: *Theoret. Comput. Sci.* **59** (1, 2) (1988) 85–114.
- [6] K.B. Bruce and G. Longo, A modest model of records, inheritance, and bounded quantification, in: *Proc. 3rd IEEE Symp. on Logic in Computer Science*, Edinburgh, Scotland (1988) 38–50.
- [7] K.B. Bruce, A.R. Meyer and J.C. Mitchell, The semantics of second-order lambda calculus, *Inform. and Comput.* to appear.
- [8] A. Carboni, P. Freyd and A. Scedrov, A categorical approach to realizability and polymorphic types, in: M. Main et al., eds., *Proc. 3rd ACM Workshop on the Mathematical Foundations of the Programming Semantics*, New Orleans, April, 1987, *Lecture Notes in Computer Science* **298** (Springer, Berlin, 1988) 23–42.
- [9] L. Cardelli, Time for a new language, Preprint, April 1988.
- [10] L. Cardelli and P. Wegner, On understanding types, data abstraction, and polymorphism, *Comput. Surveys* **17** (1985) 471–522.
- [11] R. Constable et al., *Implementing Mathematics with the Nuprl Proof Development System* (Prentice Hall, Englewood Cliffs, NJ, 1986).
- [12] T. Coquand and G. Huet, Constructions: a higher-order proof system for mechanizing mathematics, in: *Proc. EUROCAL '85, Lecture Notes in Computer Science* **203** (Springer, Berlin, 1985) 151–184.

- [13] T. Coquand, C.A. Gunter and G. Winskel, Di-domains as a model of polymorphism, in: M. Main et al., eds., *Proc. 3rd ACM Workshop on the Mathematical Foundations of the Programming Language Semantics*, New Orleans, April 1987, Lecture Notes in Computer Science **298** (Springer, Berlin, 1988) 344–363.
- [14] T. Coquand, C.A. Gunter and G. Winskel, Domain theoretical models for polymorphism, *Inform. and Comput.* to appear.
- [15] G. Cousineau, CAML, Lectures at the University of Texas Programming Institute on the Logical Foundations of Functional Programming, Austin, TX, June 1987.
- [16] E.J. Dubuc and R. Street, Dinatural transformations, in: *Reports of the Midwest Category Seminar IV*, Lecture Notes in Mathematics **137** (Springer, Berlin, 1970) 126–128.
- [17] S. Eilenberg and G.M. Kelly, A generalization of the functorial calculus, *J. Algebra* **3** (1966) 366–375.
- [18] P.J. Freyd, Structural polymorphism I, II, III (Preliminary Report), University of Pennsylvania, January 1989.
- [19] P.J. Freyd, J.Y. Girard, A. Scedrov and P.J. Scott, Semantic parametricity in polymorphic lambda calculus, in: *Proc. 3rd IEEE Symp. on Logic in Computer Science*, Edinburgh, Scotland (1988).
- [20] J.Y. Girard, Une extension de l'interprétation de Gödel, in: J.E. Fenstad, ed., *Second Scandinavian Logic Symposium, 1970* (North-Holland, Amsterdam, 1971) 63–92.
- [21] J.Y. Girard, Interprétation fonctionnelle et élimination des coupures de l'arithmétique d'ordre supérieur, Thèse de Doctorat d'Etat, Université de Paris VII, 1972.
- [22] J.Y. Girard, The system  $F$  of variables types, fifteen years later. *Theoret. Comput. Sci.* **45** (1986) 159–192.
- [23] M.J.C. Gordon, R. Milner and C. Wadsworth, *Edinburgh LCF*, Lecture Notes in Computer Science **78** (Springer, Berlin, 1979).
- [24] G. Huet, Deduction and computation, in: W. Bibel and P. Jorrand, eds., *Fundamentals of Artificial Intelligence*, Lecture Notes in Computer Science (Springer, Berlin, 1986).
- [25] G. Huet, ed., A uniform approach to type theory, in: *Logical Foundations of Functional Programming, Proceedings University of Texas Programming Institute*, Austin, TX (1987).
- [26] G. Huet, ed., *Logical Foundations of Functional Programming, Proceedings University of Texas Programming Institute*, Austin, TX (1987).
- [27] J.M.E. Hyland, The effective topos, in: A.S. Troelstra and D. van Dalen, eds., *The L.E.J. Brouwer Centenary* (North-Holland, Amsterdam, 1982).
- [28] J.M.E. Hyland, A small complete category, *Ann. Pure Appl. Logic* **40** (1988) 135–165.
- [29] J.M.E. Hyland, E.P. Robinson and G. Rosolini, The discrete objects in the effective topos, Preprint, 1987.
- [30] J.M.E. Hyland, E.P. Robinson and G. Rosolini, Algebraic types in PER models, Technical Report 88-234, Queen's University Dept. of Computing, Kingston, Ont., 1988.
- [31] M. Kelly, Many-variable function calculus I, in: S. MacLane, ed., *Coherence in Categories*, Lecture Notes in Mathematics **281** (Springer, Berlin, 1977) 66–105.
- [32] S.C. Kleene, *Introduction to Metamathematics* (Van Nostrand, New York, 1952).
- [33] J. Lambek and P.J. Scott, *Introduction to Higher-Order Categorical Logic*, Studies in Advanced Mathematics **7** (Cambridge University Press, Cambridge, 1986).
- [34] P. J. Landin, The next 700 programming languages, *Comm. ACM* **9** (1964) 157–166.
- [35] B. Liskov et al., *Clu Reference Manual*, Lecture Notes in Computer Science **114** (Springer, Berlin, 1981).
- [36] G. Longo and E. Moggi, Constructive natural deduction and its “modest” interpretation, in: J. Meseguer et al., eds., *Workshop on Semantics of Natural and Computer Languages*, Stanford, March 1987 (MIT Press, Cambridge, MA, to appear).
- [37] S. MacLane, *Categories for the Working Mathematician*, Graduate Texts in Mathematics **5** (Springer, Berlin, 1971).
- [38] P. Martin-Löf, Constructive mathematics and computer programming, in: *Sixth Internat. Congress of Logic, Methodology and Philosophy of Science* (North-Holland, Amsterdam, 1982) 153–175.
- [39] J. Meseguer, Relating models of polymorphism, Preprint, SRI International, Menlo Park, CA, U.S.A., 1988.
- [40] R. Milner, A proposal for standard ML, in: *Proc. ACM Symp. on LISP and Functional Programming* (1984) 184–197.
- [41] J.C. Mitchell, A type-inference approach to reduction properties and semantics of polymorphic expressions, in: *Proc. 1986 ACM Symp. on Lisp and Functional Programming* (1986) 308–319.

- [42] J.C. Mitchell and R. Harper The essence of ML, in: *Proc. 15th ACM Symp. on Principles of Programming Languages* (1988) 28-46.
- [43] J.C. Mitchell and A.R. Meyer, Second-order logical relations, in: R. Parikh, ed., *Lecture Notes in Computer Science* **193** (Springer, Berlin, 1985) 225-236.
- [44] A. Pitts, Polymorphism is set-theoretic, constructively, in: *Proc. Symp. on Category Theory and Computer Science*, *Lecture Notes in Computer Science* **283** (Springer, Berlin, 1987).
- [45] G.D. Plotkin, LCF considered as a programming language, *Theoret. Comput. Sci.* **5** (1977) 223-256.
- [46] J.C. Reynolds, Towards a theory of type structure, in: *Lecture Notes in Computer Science* **19** (Springer, Berlin, 1974) 408-425.
- [47] J.C. Reynolds, The essence of Algol, in: J. de Bakker and J. C. van Vliet, eds., *Algorithmic Languages*, IFIP (North-Holland, Amsterdam, 1981) 345-372.
- [48] J.C. Reynolds, Types, abstraction, and parametric polymorphism, in: R.E.A. Mason, ed., *Information Processing '83* (North-Holland, Amsterdam, 1983) 513-523.
- [49] J.C. Reynolds, Polymorphism is not set-theoretic, in: Kahn et al., *Symp. on Semantics of Data Types*, *Lecture Notes in Computer Science* **173** (Springer, Berlin, 1984).
- [50] J.C. Reynolds, Preliminary design of the programming language Forsythe, Research Report, Carnegie-Mellon University, June, 1988.
- [51] J.C. Reynolds and G.D. Plotkin, On functors expressible in the polymorphic typed lambda calculus, *Inform. and Comput.*, to appear.
- [52] E.P. Robinson, How complete is *PER*? Technical Report 88-229, Dept. of Computing & Information Science, Queen's University, Kingston, Ont., Canada, 1988.
- [53] A. Scedrov, A guide to polymorphic types, in: P. Odifreddi, ed., *Logic and Computer Science, Proc. C.I.M.E. Summer School, Montecatini Terme (June 1988)*, *Lecture Notes in Computer Science* (Springer, Berlin, to appear).
- [54] D.S. Scott, Continuous lattices, in: F. W. Lawvere, ed., *Toposes, Algebraic Geometry and Logic*, *Lecture Notes in Mathematics* **274** (Springer, Berlin, 1972) 97-136.
- [55] D.S. Scott, Data types as lattices, *SIAM J. Comput.* **5** (1976) 522-587.
- [56] D.S. Scott, Domains for denotational semantics, in: *Proc. ICALP '82*, *Lecture Notes in Computer Science* **140** (Springer, Berlin, 1982).
- [57] D.S. Scott, Realizability and domain theory, Lecture at the Amer. Math. Soc. Research Conference on Categories in Computer Science and Logic, Boulder, Colorado, June 1987.
- [58] R.A.G. Seely, Categorical semantics for higher-order polymorphic lambda calculus. *J. Symbolic Logic* **52** (1987) 969-989.
- [59] M.B. Smyth and G.D. Plotkin, The category-theoretic solution of recursive domain equations. *SIAM J. Comput.* **11** (1982) 761-783.
- [60] C. Strachey, Fundamental concepts in programming languages, *Lecture Notes*, International Summer School in Computer Programming, Copenhagen, August 1967.
- [61] A.S. Troelstra, ed., *Metamathematical Investigation of Intuitionistic Arithmetic and Analysis*, *Lecture Notes in Mathematics* **344** (Springer, Berlin, 1973).
- [62] D.A. Turner, Miranda: a non-strict functional language with polymorphic types, in: J.P. Jouannaud, ed., *Functional Programming Languages and Computer Architecture*, *Lecture Notes in Computer Science* **201** (Springer, Berlin, 1985).
- [63] N. Yoneda, On Ext and exact sequences, *J. Fac. Sci. Tokyo Ser 1*, **8** (1960) 507-526.