Effective models of polymorphism, subtyping and recursion (extended abstract)

John Mitchell * *Ramesh Viswanathan**

Department of Computer Science
Stanford University, Stanford, CA 94305
{mitchell, vramesh}@cs.stanford.edu

March 10, 1995

Abstract

We develop a class of models of polymorphism, subtyping and recursion based on a combination of traditional recursion theory and simple domain theory. A significant property of our primary model is that types are coded by natural numbers using any index of their supremum operator. This leads to a distinctive view of polymorphic functions that has many of the usual parametricity properties. It also gives a distinctive but entirely coherent interpretation of subtyping. An alternate construction points out some peculiarities of computability theory based on natural number codings. Specifically, the polymorphic fixed point is computable by a single algorithm at all types when we construct the model over untyped call-by-value lambda terms, but not when we use Gödel numbers for computable functions. This is consistent with trends away from natural numbers in the field of abstract recursion theory.

1 Introduction

The work described in this paper began with a very simple-minded objective. This was to develop an elementary class of semantic models for polymorphism, subtyping and recursion based on ideas from recursion theory instead of domain theory. Our driving intuition was that the compiler for a language such as ML compiles each expression into a bit string that could be regarded as the Gödel code for some function, pair, number, or other kind of datum, as appropriate. For each type, we therefore have a set of Gödel numbers representing the elements of the type. Since semantic models are intended to provide some basis for equational reasoning, we must also define an equivalence relation on the set of Gödel codes for each type. Our intention was therefore to define a membership predicate and equivalence relation, which together give us a so-called partial equivalence relation, for each type and take the resulting quotients of subsets of the natural numbers as the "domains" of our model.

A representative sample of recent studies using partial equivalence relations were presented at the 1990 IEEE Logic in Computer Science Conference. The first of three papers describes a special class of partial equivalence relations over the natural numbers called *extensional pers* [FRMS92], the second uses partial equivalence relations over partially-ordered domains instead of natural numbers

*Supported in part by NSF Grant CCR-9303099 and a grant from the TRW Foundation.

[AP90], and the third works in the constructive setting of the *effective topos,* which boils down to partial equivalence relations over the natural numbers [Pho90b]. Since our goal is to develop a semantic model that can be viewed as a quotient of the result of compilation, the use of pers over special partially-ordered domains does not meet our objectives. While extensional pers do give us quotients of sets of bit-strings, the standard and expected representations of basic datatypes such as natural numbers and booleans, as well as type constructors such as products, function spaces and lifting, require modification if they are to be "extensional" (in the technical sense of extensional pers). Therefore, as elaborated later in the paper, this model too does not have the basic properties we would like. Our work is close in spirit to [Pho90b], although our "bottom-up" development yields a model that seems quite different in technical detail.

We begin with the category **Per** whose objects (interpretation of types) are partial equivalence relations over the natural numbers and whose maps (interpretation of terms) are computable functions, in the usual sense of computability theory. While this category is a well-known model of polymorphism and subtyping, it is not useful for interpreting recursion. This is shown by defining a per A of partial functions and a total recursive f on A that has no fixed point. (In other words, the recursion theorem does not hold for arbitrary quotients of subsets of $\mathcal{N}$.) The counter example is based on intuition from domain theory: if we order A in the natural way, it is not "complete". This suggests that the appropriate way to extend the recursion theorem to subquotients is by borrowing ideas from domain theory. In the resulting combination of recursion-theoretic and domain-theoretic techniques, it turns out that we need far fewer ideas from domain theory than [AP90, BM92], for example. (Section 10 contains a more detailed comparison with related work.)

Using essentially the same "intrinsic order" as [AP90, Pho90b], we identify a class of "cpo-like" pers, each such per R having a computable function sup_R that gives the least upper bound of any constructible chain in R. We call these *effective cpo's;* although the name is similar, these are completely different from the effectively presented cpo's described in [GS90], for example. By a generalization of the Myhill-Shepherdson theorem from recursion theory, all computable functions between effective cpo's are continuous. If an effective cpo has a least element, then the usual domain theoretic arguments give us an effective least fixed point operator.

Serendipitously, almost all of our basic desiderata are satisfied by this construction. The natural numbers (discretely ordered) and other basic data types are easily seen to be effective cpo's. The usual product, function space and other constructions preserve the property of being an effective cpo. Moreover, the intrinsic orderings turn out to be the usual "pointwise orderings" of domain theory. Since it is easy to use the supremum operator to compute a least fixed point, the full subcategory **ECpo** of effective cpo's and computable functions gives us an elementary semantics of first-order type operators and recursion.

In the model using Gödel numbers of partial recursive functions, the interpretation of higher-order types requires that we code types as natural numbers. The traditional approach, when we are not concerned with recursion, is to make the coding relation trivial, leading to an interpretation of polymorphic types as intersection of infinite indexed families. However, the intersection of a constructive family of effective cpo's is not necessarily an effective cpo. We therefore represent a type by any Gödel code of its supremum operator, producing an interpretation of a polymorphic type $\forall t.A(t)$ as the collection of recursive functions mapping any supremum operator sup_R to an element of the per $A(R)$. Fortunately, this interpretation of polymorphism still satisfies the expected parametricity principles. For example, we can show that the per $\forall t.t$ is empty and that the only element of the per $\forall t.t \rightarrow t$ is the identity function. We can also define a $\forall^p$ quantifier where we interpret the domain of quantification to be pointed effective cpos. The difference from the $\forall$ quantifier arises because the code for a pointed effective cpo naturally gives both its least element and its supremum operator. Continuing our examination of parametricity properties, we show that the least fixed point operator is the only fixed point operator of type $\forall^p t.(t \to t) \to t$. Due to least elements, the type $\forall^p t.t$ becomes nonempty and $\forall^p t.(t \to t)$ contains more than the identity function. In short, the $\forall$ quantifier supports the parametricity principles of the system **F** while the $\forall^p$ quantifier supports the parametricity principles of system **F** extended with fixed points.

A standard method for obtaining recursive types, described in [SP82], involves showing that suitable functors on **Cpo**-enriched categories have initial F-algebras. To obtain analogous results, we work with **ECpo**-enriched categories whose homsets each form an effective cpo. In addition, and in a departure from applications of the Smyth-Plotkin method we have seen before, we ask for functors that are effective on objects in addition to morphisms. This means that a functor F on **ECpo** is *effective* if the supremum function on $F(R)$ is uniformly computable from the supremum function on R. If each homset is also pointed, we have a **PECpo**-enriched category. As for morphisms in **Per**, we can show that any effective functor is automatically locally continuous (in the sense of [SP82]) and thus any effective functor between *effectively complete* **PECpo**-enriched categories has an initial F-algebra. Our categories **ECpo** and **PECpo** can be shown to be effectively complete, as well as **ECpo**-enriched and **PECpo**-enriched, respectively. This gives us initial F-algebras for all effective functors. All of this feels relatively routine, once we have the idea of realizers for effective cpos; we are simply working out effective versions of the usual arguments.

In other per models, the interpretation of subtyping has always been the subset relation on pers, namely, R is a subtype of S iff the membership predicate and equality relations of R are subsets of those for S. Intuitively, this means that a supertype may have more elements than a subtype, but also distinct elements of a subtype may become identified at a supertype. (It is argued elsewhere that this is very intuitive and natural.) In our model, we cannot simply follow this approach since we need coherence between the supremum operators at subtype and supertype, and between least elements if both are pointed. More specifically, we say an effective cpo R is a subtype of effective cpo S if in addition to being a subset, the least upper bound of any increasing chain in R is also a least upper bound in S. Moreover, if R is pointed then any index of the least element of R must also be one for S. All the usual examples of subtyping hold with our modified definition of subtyping. We also prove a strong form of coherence, namely, if two terms of the same type yield the same untyped term when types are erased, then their meanings are equal.

The interpretation is extended to higher-order polymorphism and recursion in Section 7, where we use the category of doubles of **ECpo**-enriched categories and symmetric functors.

We started off with the intent of forming an extensional model by taking quotients of sets of bit strings (or Gödel numbers of functions). One reason for doing this was to obtain a close correspondence with compilation, since the denotation of an expression could be viewed as the equivalence class of the bit string produced by compiling this expression. However, a compiler for the kind of language we are interested in may generate code without using type information. A reasonable compiler could parse expressions, annotate the parse trees during type checking, but then discard or ignore the type annotations during code generation. A consequence is that we expected to find a uniform, type-independent algorithm computing the fixed point operator at each type. It is one of the most surprising aspects of our effort that we were unable to accomplish this. Specifically, in Section 8, we use computability arguments to show that there is no uniform supremum operator and that no untyped fixed-point operator can be the index of the typed fixed-point operator at each type. The proof uses algorithms that apply natural number operations such as equality test to the Gödel codes of function arguments. This is allowed in per structures as long as the resulting functional has the right extensional behavior.

In response to our difficulties with natural number codes, Gordon Plotkin suggested using untyped call-by-value lambda terms instead. The result, described in Section 9, is a very similar category of effective cpo's, but with a single untyped term computing all fixed-points. This suggests that compilation is more profitably viewed as a translation to untyped lambda terms. While we initially regarded bit strings as the result of compilation, we realize upon reflection that the result of compiling a typed program is always a "typed algorithm." In other words, although machine language allows us to do bit string operations on function code, for example, no such manipulation need occur in the object code produced by compiling a typed program. Since natural numbers are distinct from function expressions in the untyped call-by-value calculus, there is just enough of an inherent type distinction to provide a uniform fixed-point algorithm. While this may be a slight surprise to computer scientists, our eventual preference for call-by-value terms appears consistent with the move away from natural number coding in the field of recursion theory (e.g., [Bar75]).

2 Preliminaries

Let $\varphi_0, \varphi_1, \ldots$ be any enumeration of the partial recursive functions. We use $n \cdot m$ to denote $\varphi_n(m)$. We use *pr* for a computable pairing function on the natural numbers and $(n)_1$ and $(n)_2$ to denote the first and second projections of n. Let $\mathcal{N}$ denote the set of natural numbers. A partial equivalence relation (per) $R \subseteq \mathcal{N} \times \mathcal{N}$ is a symmetric transitive relation. For any per R, $n{:}\,R$ denotes that $n\ R\ n$; for any $n{:}\,R$, $[n]_R$ denotes the equivalence class $\{m \mid n\ R\ m\}$ and the set represented by R is defined to be $[R] = \{[n]_R \mid nRn\}$. We say that a natural number n realizes an element $a \in [R]$, $n \vdash_R a$, if $n \in a$. A function f (set-theoretic) from $[R]$ to $[S]$ is realized by a natural number n, denoted by $n \vdash_{R,S} f$ if for all $a \in [R]$, $m \vdash_R a$ we have that $n \cdot m \vdash_S f(a)$ and we call a function $f{:}\,[R] \rightarrow [S]$ effective or computable if it has a realizer. We often omit the subscripts R, S from $n \vdash_{R,S} f$ when they are clear from the context. We use $[n]^{R,S}$ to denote the unique map from $[R]$ to $[S]$ that n realizes, if it realizes one. The category **Per** has as objects pers, and as morphisms from R to S the effective maps from $[R]$ to $[S]$. For any set $B \subseteq \mathcal{N}$, we define the per $D_B = \{\langle n, n\rangle \mid n \in B\}$, the identity relation on B. We use N to denote the per $D_{\mathcal{N}}$ and $\mathbf{1}$ to denote the per $D_{\{0\}}$. If R, S are pers, we define the pers $R \times S, R \rightarrow S, R \rightharpoonup S$ by $n\, R \times S\, m$ iff $(n)_1\, R\, (m)_1$ and $(n)_2\, S\, (m)_2$, $n\, R \rightarrow S\, m$ iff $\forall x, y \in \mathcal{N}.\ x\, R\, y \Rightarrow (n \cdot x)\ S\ (m \cdot y)$, $n\ R \rightharpoonup S\ m$ iff $\forall x, y \in \mathcal{N}.\ x\ R\ y \Rightarrow (n \cdot x) \downarrow \Rightarrow (n \cdot x)\ S\ (m \cdot y)$ where we use $(n \cdot x)\ S\ (m \cdot y)$ to mean that $(n \cdot x) \downarrow$ and $(m \cdot y) \downarrow$ and they are related in S. We can define the lifting of a per R as $R_\perp = \mathbf{1} \rightharpoonup R$.

Let A be any set and $\vdash_A \subseteq \mathcal{N} \times A$ be a binary relation which we often call a realization relation. Recall that $\vdash_A$ is *onto* if for all $a \in A$, there is an n with $n \vdash_A a$ and a *function* if whenever $n \vdash_A a, n \vdash_A b$ we have that $a = b$. Every onto function $\vdash_A$ on a set A gives rise to a per, namely, $\{\langle n, m\rangle \mid n, m \vdash_A a,\ a \in A\}$. Suppose $\vdash_A$ is some fixed onto realization relation on a set A and $\vdash_B$ is another fixed onto realization relation on a set B. We call a function (set-theoretic) $f{:}\,A \rightarrow B$ *effective* if there is an n such that for all $a \in A, m \vdash_A a$ we have that $n \cdot m \vdash_B f(a)$; we take n to be the realizer of f. We also say that $f(a)$ is *effectively realizable* from a to mean that the function f is effective.

3 Fixed Points and Effective Cpos

We begin by showing that the full category **Per** is not suitable for interpreting fixed points.

Example: [Failure of Fixed Points] Take R to be the per $\mathbf{1}$ and S to be the per of partial functions that are defined exactly on some finite prefix of the natural numbers. More formally, S is defined by

$m\,S\,n$ iff $\exists u \in \mathcal{N}\ \forall x < u\ (m \cdot x = n \cdot x)$ and $\forall x \geq u\ (m \cdot x\uparrow$ and $n \cdot x\uparrow)$. Let F be an index of the function, which on input $f\colon (R \rightharpoonup S)$, returns an index of the function, which on input $0\colon \mathbf{1}$ returns an index of the function which on input n computes $g = f \cdot 0$ and returns 0 if $n = 0$ and returns $g \cdot (n-1)$ otherwise. It can be verified that $F\colon (R \rightharpoonup S) \to (R \rightharpoonup S)$ and that $[F]^{(R \rightharpoonup S),(R \rightharpoonup S)}$ has no fixed point.

An important idea in the above example is that if we order the partial functions that are the elements of the per S by the pointwise ordering, as we would in domain theory, then S is not a cpo. Therefore, it seems natural to circumvent the problem illustrated by the per S by selecting a class of cpo-like pers. The first step is to associate an order with each per. To define an ordering on the elements of a per R, we focus attention on effective partial functions from R to $\mathbf{1}$. Since, $\mathbf{1}$ is a one-element per, the only significant behaviour of any $f\colon R \rightharpoonup \mathbf{1}$ on an $x\colon R$ is its convergence. We can thus think of $f\colon R \rightharpoonup \mathbf{1}$ as a partial decision algorithm computing with R-inputs or a *computable test* on R, with convergence announcing success. We then consider an $x\colon R$ to be distinguishable from $y\colon R$ if there is a computable test converging on x and diverging on y. Thus, an $x\colon R$ has computationally no more information than $y\colon R$ if x is not computationally distinguishable from y. Thus, for any per R and $x, y\colon R$, we define

$$x \leq_R y \ \text{ iff } \ \forall f\colon R \rightharpoonup \mathbf{1}.\ f \cdot x\downarrow \ \text{ implies } \ f \cdot y\downarrow$$

For $[x]_R, [y]_R \in [R]$, we define $[x]_R \leq_{[R]} [y]_R$ iff $x \leq_R y$; $\leq_{[R]}$ is easily seen to be a preorder.

Taking this to be our *intrinsic ordering*, we consider the class of pers whose intrinsic orderings are complete partial orders in the internal language of the topos **Per**. For any per R, define the per $\mathit{Seq}(R)$ of *effective increasing sequences* as the largest subset of the per $N \to R$ such that $s\colon \mathit{Seq}(R)$ iff for all $n \in \mathcal{N}$, $s \cdot n \leq_R s \cdot (n+1)$. We call a function (possibly partial) from $[\mathit{Seq}(R)]$ to $[R]$ a *supremum* function if it maps an element $[s]$ of $[\mathit{Seq}(R)]$ to a least upper bound of $\{[s \cdot n]_R \mid n \in \mathcal{N}\}$ with respect to the preorder $\leq_{[R]}$ if it exists and undefined otherwise. If the preorder $\leq_{[R]}$ on $[R]$ is antisymmetric, then there is a unique supremum function; in this case, we denote it by sup_R. We define a per R to be an *effective cpo* if

Antisymmetry For $a, b \in [R]$, if $a \leq_{[R]} b$ and $b \leq_{[R]} a$ then $a = b$.

Computable Completeness The supremum function $\mathit{sup}_R\colon [\mathit{Seq}(R)] \to [R]$ is a morphism in **Per**, *i.e.*, it is *total* and *effective.*

Lemma 3.1 (Continuity) *Suppose that R, S are effective cpos. Then any effective $f\colon [R] \to [S]$ is continuous, i.e., if $s\colon \mathit{Seq}(R)$ then $f(\mathit{sup}_R([s]_{\mathit{Seq}(R)})) = \mathit{sup}_S(f \circ [s]^{N,R})$.*

By analogy with ordinary cpos, we say that an effective cpo R is *pointed* if $[R]$ contains a least element with respect to the intrinsic order $\leq_{[R]}$; we denote its least equivalence class by $\perp_R$.

Lemma 3.2 (Fixed Points) *Suppose that R is a pointed effective cpo. Every effective $f\colon [R] \to [R]$ has a least fixed point with respect to the ordering $\leq_{[R]}$. The function $\mathit{fix}_R\colon [R \to R] \to [R]$ mapping realizers for morphisms to their least fixed point is effective and is effectively realizable from sup_R and $\perp_R$.*

While our assumptions on the properties of an effective cpos were important to our proof of the existence of a computable fixed point operator, we can also show that they are the weakest conditions possible. We can construct a per R whose intrinsic order is not antisymmetric, and a per S whose intrinsic order is antisymmetric and for which sup_S is total but not effective; both R

and S can be shown not to admit any effective fixed point operator. It can also be seen that a per that is not pointed cannot admit a fixed point operator.

The basic constructions on pers all yield effective cpos. For any $B \subseteq \mathcal{N}$, the intrinsic order on the per D_B is easily seen to be discrete and hence an effective cpo trivially and it is pointed iff $|B| = 1$; we now state the closure under the other constructions.

Proposition 3.3 (Product Spaces) *If R, S are pers then $x \leq_{R\times S} y$ iff $(x)_1 \leq_R (y)_1$ and $(x)_2 \leq_S (y)_2$. Thus, if R and S are effective cpos then $R \times S$ is an effective cpo, with $sup_{R\times S}$ effectively realizable from sup_R and sup_S. If R, S are pointed then $R \times S$ is pointed with $\bot_{R\times S}$ effectively realizable from $\bot_R, \bot_S$.*

Proposition 3.4 (Total Function Spaces) *If R, S are pers and S is an effective cpo, then $f \leq_{R\to S} g$ iff $\forall x{:}\, A. f \cdot x \leq_S g \cdot x$. Thus, if R and S are effective cpos then $R \to S$ is an effective cpo, with $sup_{R\to S}$ effectively realizable from sup_S. If S is pointed then $R \to S$ is pointed with $\bot_{R\to S}$ effectively realizable from $\bot_S$.*

Proposition 3.5 (Partial Function Spaces) *If R, S are pers and S is an effective cpo, then $f \leq_{R\rightharpoonup S} g$ iff $\forall x{:}\, A. f \cdot x \downarrow \Rightarrow f \cdot x \leq_S g \cdot x$ Thus, if R and S are effective cpos then $R \rightharpoonup S$ is an effective cpo, with $sup_{R\rightharpoonup S}$ computable from sup_S. The per $R \rightharpoonup S$ is always pointed with $\bot_{R\rightharpoonup S}$ realized by the index of the function "$\lambda x \in \mathcal{N}.diverge$" independent of R, S.*

While the proof of the "pointwise" intrinsic ordering for products is standard (*c.f.* [AP90]) and is true for arbitrary pers, the intrinsic ordering for the pers $R \to S, R \rightharpoonup S$ is pointwise only when the per S is an effective cpo. An important point to note about the closure of effective cpos under these operations, as given by the above propositions, is that it is constructive, *i.e.*, the supremum operator for the product or function space pers of R, S can be computed from the supremum operator for R and S. Since lifting is defined using partial functions, it follows that effective cpos are closed under lifting as well.

4 Polymorphism

Let ECpo and PECpo be the set of all effective cpos and pointed effective cpos respectively. Define an *effective set* to be a set K equipped with an *onto* binary relation $\vdash_K \subseteq \mathcal{N} \times K$. For any function $\mathcal{C}{:}\, K \to \mathrm{ECpo}$, we can define the per $\forall^K(\mathcal{C})$ as follows:

$$f \; \forall^K(\mathcal{C}) \; g \quad \text{iff} \quad \forall a \in K, x, y \in \mathcal{N} \quad x, y \vdash_K a \Rightarrow (f \cdot x) \; \mathcal{C}(a) \; (g \cdot y)$$

It is easy to see that $\forall^K(\mathcal{C})$ is symmetric and transitive, *i.e.*, that it is a per. We take ECpo to be an effective set with $n \vdash_{\mathrm{ECpo}} R$ iff $n \vdash_{Seq(R),R} sup_R$. And, we consider PECpo an effective set with $n \vdash_{\mathrm{PECpo}} R$ iff $(n)_1 \vdash_{\mathrm{ECpo}} R$ and $(n)_2 \vdash_R \bot_R$. Essentially the realizers for ECpo and PECpo are proofs, in the internal language of the effective topos, of the membership of a per in the particular subset. The following lemma shows that if $\mathcal{C}$ is an effective map (as defined in Section 2) from K to ECpo then $\forall^K(\mathcal{C})$ is an effective cpo. As usual, we also give the conditions under which it is pointed.

Lemma 4.1 (Higher-Order Polymorphism) *Let $\mathcal{C}{:}\, K \to \mathrm{ECpo}$ be an effective map. Then $f \leq_{\forall^K(\mathcal{C})} g$ iff $\forall a \in K, x \in \mathcal{N} \quad x \vdash_K a \Rightarrow (f \cdot x) \leq_{\mathcal{C}(a)} (g \cdot x)$. Thus, $\forall^K(\mathcal{C})$ is an effective cpo with $sup_{\forall^K(\mathcal{C})}$ effectively realizable from $\mathcal{C}$. If $\mathcal{C}$ induces an effective map from K to PECpo then $\forall^K(\mathcal{C})$ is pointed with $\bot_{\forall^K(\mathcal{C})}$ effectively realizable from the effective map $\mathcal{C}{:}\, K \to \mathrm{PECpo}$.*

As particular instances of Lemma 4.1, we can define constructors $\forall^{\mathrm{ECpo}}$ which we denote $\forall$, and $\forall^{\mathrm{PECpo}}$ which we denote $\forall^p$, yielding effective cpos. The following proposition shows that in this semantics of polymorphism as well, the type $\forall t.t$ is empty and $\forall t.(t \to t)$ only contains the identity function. It thus suggests that all the usual parametricity principles are validated even by this interpretation of polymorphism that is less "uniform" than intersection. Essentially, parametricity in our context arises from the fact that the realization relations $\vdash_{\mathrm{ECpo}}, \vdash_{\mathrm{PECpo}}$ are *not functions* — thus, any element of the polymorphic per must behave uniformly on all pers that admit common realizers for their supremum functions.

Proposition 4.2 (Parametricity) *For $\mathcal{I}$: ECpo $\to$ ECpo defined by $\mathcal{I}(R) = R$, there is no n with $n{:}\forall(\mathcal{I})$. For $\mathcal{F}$: ECpo $\to$ ECpo defined by $\mathcal{F}(R) = R \to R$, if $n{:}\forall(\mathcal{F})$ then for any effective cpo R, natural number x, if $x \vdash sup_R$ then $n \cdot x \vdash_R id_R$.*

Turning next to the $\forall^p$ quantifier, consider the map $\mathcal{C}{:}\,\mathrm{PECpo} \to \mathrm{ECpo}$ given by $\mathcal{C}(R) = (R \to R) \to R$ which is effective by Proposition 3.4. By Lemma 3.2 we have a realizer $\mathit{fix}{:}\forall^p(\mathcal{C})$, which computes the least fixed point at every pointed type. The following proposition shows that this is the *only* fixed point operator of this polymorphic type:

Proposition 4.3 (Polymorphic Fixed Point) *Consider the effective map $\mathcal{C}{:}\,\mathrm{PECpo} \to \mathrm{ECpo}$ given by $\mathcal{C}(R) = (R \to R) \to R$. Then we have a realizer $\mathit{fix}{:}\forall^p(\mathcal{C})$ computing the least fixed point at every pointed type. Further, suppose that we have an $f{:}\forall^p(\mathcal{C})$ such that for any pointed effective cpo R, and $x \vdash sup_R, b \vdash \bot_R$ we have that $(f \cdot pr(x, b)) = n$ such that for any $g{:}\, R \to R$, $n \cdot g\ R\ g \cdot (n \cdot g)$,* i.e.*, f is a fixed point operator. Then $f\ \forall^p(\mathcal{C})\ \mathit{fix}$.*

Just like Proposition 4.2, we can prove that the type $\forall^p t.t$ has exactly one element and the type $\forall^p t.(t \to t)$ has exactly two elements; the polymorphic operator $\forall^p$ thus supports the parametricity principles of system **F** extended with recursion on values.

5 Recursive Types

A standard method for obtaining recursive types is that given by [SP82] which shows that suitable functors on **Cpo**-enriched categories admit recursive solutions. While these results are not directly applicable to the category of effective cpos, we obtain analogous results by considering everything "effectively" and reformulating the framework and results of [SP82] for a suitable notion of effectivity. The main departure from earlier work in this regard (*e.g.*, [AP90]) is that effectiveness of a functor is a condition on its behaviour on objects in addition to arrows.

Let $\mathbf{C}$ be a category. We call it *effective* if it is equipped with a realization relation, $\vdash_{\mathbf{C}}$, on objects and morphisms that is *onto on objects* and an *onto function on each homset*, with identity morphisms effectively realizable from the objects and composition effective; define effective functors between effective categories in the obvious way. By the condition on $\vdash_{\mathbf{C}}$, we can define a per corresponding to each homset as in Section 2. We say that $\mathbf{C}$ is an **ECpo**-enriched category if the per corresponding to each homset $\mathbf{C}(A, B)$ is an effective cpo with its supremum function effectively realizable from the objects A, B; this is of course just the internal proof that every homset is a cpo. Define the category $\mathbf{C}^{\mathrm{ep}}$ whose objects are those of $\mathbf{C}$ and whose morphisms are embedding-projection pairs (while the embeddings and projections determine each other uniquely, they may not necessarily do so effectively; it therefore does not suffice to consider the subcategory of only embeddings or projections). As for morphisms in **Per**, any effective functor between two **Ecpo**-enriched categories is automatically locally continuous in the sense of [SP82]. Call an effective category $\mathbf{C}$ a **PECpo**-enriched category if it is an **ECpo**-enriched category and the per corresponding to each homset $\mathbf{C}(A, B)$ is pointed with its least element effectively realizable from the objects A, B.

Lemma 5.1 (Recursive Solutions) *Suppose that* $\mathbf{C}$ *is an effectively* $\boldsymbol{\omega}^{\mathrm{op}}$*-complete,* **PECpo***-enriched category with a terminal object* $\perp$ *and such that composition in* $\mathbf{C}$ *is left-strict,* i.e., *for any* $A \xrightarrow{f} B$ *we have* $\perp_{B,C} \circ f = \perp_{A,C}$. *Then for any effective functor* $F\colon \mathbf{C}^{\mathrm{ep}} \to \mathbf{C}^{\mathrm{ep}}$, *we have an object* $\mu(F)$ *in* $\mathbf{C}$ *and morphisms* $\mathit{fold}^F\colon F(\mu(F)) \to \mu(F)$, $\mathit{unfold}^F\colon \mu(F) \to F(\mu(F))$ *with* $\mathit{fold}^F, \mathit{unfold}^F$ *constituting an isomorphism pair in* $\mathbf{C}$. *Moreover, the object* $\mu(F)$ *and the morphisms* $\mathit{fold}^F, \mathit{unfold}^F$ *are effectively realizable from the effective functor* F.

Define **ECpo** to be the full subcategory of **Per**, of the effective cpos. The realization relation on objects of **ECpo** is as in Section 4 for ECpo and on morphisms as in Section 2. Take **PECpo** to be the full subcategory of **ECpo** of pointed effective cpos with the realization relation on objects as in Section 4 for PECpo.

Lemma 5.2 *The categories* **ECPo** *and* **PECpo** *are effectively* $\boldsymbol{\omega}^{\mathrm{op}}$*-complete.*

By Proposition 3.4, **ECpo** is an **ECpo**-enriched category and **PECpo** is a **PECpo**-enriched category; it can also be seen that **PECpo** satisfies the other conditions of Lemma 5.1. Hence, any effective functor $F\colon \mathbf{ECpo}^{\mathrm{ep}} \to \mathbf{ECpo}^{\mathrm{ep}}$ that induces an effective functor from $\mathbf{PECpo}^{\mathrm{ep}}$ to $\mathbf{PECpo}^{\mathrm{ep}}$ has a recursive solution. This condition on the preservation of pointedness is not an accident of our method of constructing recursive solutions; there are functors that do not map pointed pers to pointed pers and provably cannot have any recursive solutions.

6 Subtyping

Suppose we have pers R, S with $R \subseteq S$ as relations. Since the index of $\lambda x \in \mathcal{N}.\, x$ realizes an effective map from R to S, we have by Lemma 3.1, that if $x \leq_R y$ then $x \leq_S y$. Thus, if $R \subseteq S$, then $\mathit{Seq}(R) \subseteq \mathit{Seq}(S)$. For effective cpos R, S, we define that R is a subtype of S, denoted $R <: S$, iff $R \subseteq S$, if $n \vdash \mathit{sup}_R$ and $s\colon \mathit{Seq}(R)$ then $n \cdot s \vdash \mathit{sup}_S(s)$, and if R is pointed and $n \vdash_R \perp_R$ then S is pointed with $n \vdash_S \perp_S$. Essentially then, R is a subtype of S, if the proof of any "interesting property" in R is also a proof in S; the interesting properties in our context being membership in a per, equality in a per, and membership in the subcategories **ECpo**, **PECpo**. This notion of subtyping is reflexive and transitive and one obtains all the usual subtyping relations between the various type constructors. We can define the per $\mathit{Top} = \mathcal{N} \times \mathcal{N}$ as the supertype of all types.

7 A Model of $F\omega$ with recursion and subtyping

We can present our semantics as a categorical version of [BMM90] with additional structure for the distinction between types and pointed types, fixed points at pointed types, higher-order polymorphism and recursive solutions for maps from pointed kinds to pointed kinds. At a first approximation, we can take the category for kinds to be the category of all **ECpo**-enriched categories and effective functors, and take the categories **ECpo** and **PECpo** to interpret types and pointed types respectively. The problem with this is that type operators such as $\to$ cannot be expressed as covariant functors and are thus not morphisms of this category. Inspired by [AP90], we consider doubles of categories and symmetric functors; our main technical contribution is that we exhibit a cartesian closed structure, which allows us to give a categorical semantics of $\mathbf{F}\omega$ and recursive kinds.

Let $\mathcal{K}$ be the category whose objects are $\mathbf{C}^D$ for $\mathbf{C}$ an effectively $\boldsymbol{\omega}^{\mathrm{op}}$-complete, **ECpo**-enriched category with terminal object and left-strict composition (which categories we henceforth abbreviate as "good" categories), and whose morphisms are effective symmetric functors. We can define the product of $\mathbf{C}^D, \mathbf{D}^D$ itself as the double of another "good" category and similarly for exponentials; this establishes $\mathcal{K}$ to be cartesian-closed. We take types to be the object $\mathbf{ECpo}^D$ and pointed types to be $\mathbf{PECpo}^D$. By extending the definitions of all the semantic constructions given in earlier sections on objects to morphisms in the standard way, we can define them as symmetric effective functors into $\mathbf{ECpo}^D$ as well as $\mathbf{PECpo}^D$ describing their behaviour as type constructors as well as pointed type constructors.

We interpret terms as effective natural transformations between appropriate functors (essentially, dropping the "double" on their domain and range). The term constructors arise from the adjunctions expressing the cartesian closure of **ECpo**, the natural transformations asserting that the functor $(\cdot)_\perp$ is a monad, the natural isomorphism between $R \rightharpoonup S$ and $R \rightarrow S_\perp$, and $\forall^{\mathbf{C}}$ as a right adjoint to the diagonal functor. Finally, by Lemma 3.2 the morphism fix^R is effectively realizable from R and by Lemma 5.1 the morphisms $\mathit{fold}^F, \mathit{unfold}^F$ are effectively realizable from F; we can also prove that they are natural in R and F respectively which gives their interpretations as effective natural transformations. It turns out that except for the fixed point operator and the isomorphism pair between a recursive kind and its unfolding, every other term constructor arises from a *uniform* natural transformation.

We define the subtyping relation for objects of **ECpo** as in Section 6. Usually, coherence of a semantics (in the presence of subtyping) means that the meaning of a term does not depend on its typing derivation. Inspite of a fixed point operator whose interpretation depends on the type, we can prove a stronger form of coherence for our semantics. Informally, we can show that if for two terms M, N of type σ their erasures are the same then $[\![M{:}\,\sigma]\!] = [\![N{:}\,\sigma]\!]$, *i.e.*, their meanings are the same at type σ. We show this property for fixed points and the isomorphism pair between a recursive type and its unfolding; since the interpretation of all other term constructors arises from uniform natural transformations, our coherence theorem asserting the equality of the meaning of two terms with the same erasure at a particular type follows.

8 Uniform Fixed Point Realizer

We first investigate the existence of a uniform realizer for all supremum functions. The following lemma shows that this is impossible even for $N_\perp, (N \rightharpoonup N)_\perp$.

Lemma 8.1 (Failure of Uniform Suprema) *There is no natural number f such that $f \vdash sup_N$ and $f \vdash sup_{N \rightharpoonup N}$,* i.e.*, such that if $s{:}\,Seq(N)$ then $f \cdot s \vdash_N sup_N(s)$ and if $s{:}\,Seq(N \rightharpoonup N)$ then $f \cdot s \vdash_{N \rightharpoonup N} sup_{N \rightharpoonup N}(s)$.*

Corollary 8.2 *There is no natural number f such that $f \vdash sup_{N_\perp}$ and $f \vdash sup_{(N \rightharpoonup N)_\perp}$.*

Next, we focus on uniformly computing fixed points of $f{:}\,(R \rightharpoonup S) \rightarrow (R \rightharpoonup S)$ for effective cpos R, S. A natural candidate for a uniform fixed point operator is the index, Y, of the untyped call-by-value fixed point operator (also given by the first recursion theorem *e.g.*, see [Cut80]). Of course, untyped lambda calculus abounds with other fixed point operators (*c.f.* [Bar84]); the following definition captures the essential property of any untyped fixed point operator.

Definition 8.3 A natural number f is an *untyped fixed point operator* if for any $n: N \to N$ we have $f \cdot n\ (N \rightharpoonup N)\ n \cdot (f \cdot n)$.

Lemma 8.4 (Failure of Untyped Fixed Points) *Consider any untyped fixed point operator f. Suppose that R, S are pers with $\neg(k(R \rightharpoonup S)k)$, for some $k \in \mathcal{N}$. Then there is an $e: (R \rightharpoonup S) \to (R \rightharpoonup S)$ with $\neg(f \cdot e(R \rightharpoonup S) f \cdot e)$. Thus, f does not even realize a morphism from $((R \rightharpoonup S) \to (R \rightharpoonup S))$ to $(R \rightharpoonup S)$ and hence cannot realize a fixed point operator for the per $R \rightharpoonup S$.*

In particular, the index Y cannot be a realizer for the fixed point operator on all $R \rightharpoonup S$. The definition of e, in the proof of Lemma 8.4, uses an equality test on its argument y which essentially depends on the fact that realizers for functions ($y: R \rightharpoonup S$ in this case) are accessible as realizers of natural numbers and thus susceptible to equality tests. In Section 9 we show that this is the only reason for the failure of Y as a fixed point operator.

9 Effective Cpos over Lambda Terms

We consider an untyped call-by-value λ-calculus that has term constants corresponding to numerals and some basic operations on natural numbers. It thus corresponds closely to the "turing machine" application operation on Gödel numbers, except for introducing a distinction between codes of natural numbers (numerals) and codes of functions (λ-abstractions). We take the partial combinatory algebra whose elements are observational congruence classes of the terms of the calculus and consider effective cpos over them. We inherit all the results from previous sections on effective cpos over natural numbers, since their proofs only use computability arguments, and all computable functions are expressible in this calculus.

We can show that the untyped term $Y = \lambda f.\, (\lambda x.\, f\ (\lambda z.\, xxz))\ (\lambda x.\, f\ (\lambda z.\, xxz))$ is a realizer for the least fixed point operator $\mathit{fix}^{R \rightharpoonup S}$ for any effective cpos R, S, using the observational preorder $\sqsubseteq$ on terms. The partial combinatory algebra, equipped with this ordering, is not a cpo; however, if $a_n, n \in \mathcal{N}$ are such that $a_n \sqsubseteq a_{n+1}$, we denote its least upperbound by $\bigsqcup_n a_n$ when it exists. For any effective cpo R, we can prove the remarkable property that if $s: \mathit{Seq}(R)$ is an increasing sequence such that $s \cdot n \sqsubseteq s \cdot (n+1)$ then $\bigsqcup_n (s \cdot n) \vdash_R \mathit{sup}_R(s)$, *i.e.*, if an increasing sequence is also an increasing sequence with respect to $\sqsubseteq$ then its supremum in the effective cpo must be the equivalence class of the least upper bound with respect to the global ordering. Now, consider the elements F_n defined by $F_0 = \lambda f.\, \Omega$, where Ω is any divergent term and $F_{n+1} = \lambda f.\, f\ (\lambda z.\, F_n\ f\ z)$. Because of the distinction between numerals and λ-abstractions, $F_n \sqsubseteq F_{n+1}$ and $Y = \bigsqcup_n F_n$. We also have that $F_n: ((R \rightharpoonup S) \to (R \rightharpoonup S)) \to (R \rightharpoonup S)$ and that $\mathit{fix}^{R \rightharpoonup S} = \mathit{sup}_{((R \rightharpoonup S) \to (R \rightharpoonup S)) \to (R \rightharpoonup S)}(s)$ where s is the sequence with $s \cdot n = F_n$. From all this it follows that $Y \vdash \mathit{fix}^{R \rightharpoonup S}$.

10 Comparison with related work

From domain theory, we have several classes of domain models of polymorphism and recursion. The first are the universal domains. A general construction explored in [ABL86] interprets types as the finitary projections over a universal domain. Since types are domains, recursion is straightforward. An alternative model in essentially the same spirit uses the closures of $P\omega$ [Sco76, McC79, BMM90]. Two models that are not based on universal domains are the coherent spaces of Girard [Gir86] and the related use of dI-domains developed in [CGW89]. In each of these cases, we would interpret subtyping by selecting a class of continuous "subtype maps." However, since there is no general characterization of subtyping, this would have to be done on an *ad hoc* basis for each model.

An example study identifying the linear maps between dI-domains as candidate subtype maps is [BTCGS91].

The alternatives to these domain models all use partial equivalence relations. The per models fall into two groups, one using pers over domains and the other pers over the natural numbers. Some constructions in the first group are explained in [AP90, Ama91, Car89, BM92]. The most sophisticated construction is [AP90], which uses an intrinsic order similar to the present paper, but in addition to being a cpo, the pers need to satisfy other conditions (*e.g.*, uniformity). In as much as the model presented in Section 9 works with only the simple conditions of being an effective cpo and is a semantics of polymorphism and recursion with uniform fixed points, these additional conditions on pers seem to be a direct consequence of working over specially constructed domains.

There are two previous models using pers over the natural numbers. The more accessible is the class of extensional pers [FRMS92]. Unfortunately, the property of being an extensional per is not categorical as evidenced by the fact that the class of extensional pers is not closed under recursive isomorphism. More importantly from our point of view, the natural interpretations of basic datatypes such as natural numbers and booleans also fail to be extensional (in the technical sense of "extensional pers"). The extensional pers are also not closed under the natural constructions of products, total function spaces and partial function spaces. Hence one uses variants of these operations that work but seem to have no natural explanation. In short, although the development of extensional pers is mathematically elegant and rich, these pers do not support our motivating intuition for preferring per models over the natural numbers.

The final class of models involve the development of domains within the effective topos [Hyl82, Hyl88]. The main idea, as explained briefly in [Pho90b] and elaborated in [Pho90a, Hyl90, HRR90] is to carry out the usual development of domain theory within a constructive set theory that may be interpreted over pers. This leads to a class of domain-like pers that are essentially the same as our effective cpos. However, the class of maps, interpretation of lifting, and fixed-point operators are technically different from ours. In particular, it does not seem that polymorphism is interpreted by considering supremum operators as the realizers of domains. There is also a model over untyped call-by-name lambda-terms in [Pho90a], using pers that have an order property defined using a specific per ω and its completion $\bar{\omega}$. The main difference is that our definition of effective cpo, which implies Phoa's condition on relations, makes sense for any partial combinatory algebra, while Phoa's condition appears specific to lambda terms.

References

[ABL86] R.M. Amadio, K. Bruce, and G. Longo. The finitary projection model for second order lambda calculus and solutions to higher order domain equations. In *Proc. IEEE Symp. on Logic in Computer Science*, pages 122–130, 1986.

[Ama91] R.M. Amadio. Recursion over realizability structures. *Information and Computation*, 91(1):55–86, 1991.

[AP90] M Abadi and G.D. Plotkin. A PER model of polymorphism and recursive types. In *Proc. IEEE Symp. on Logic in Computer Science*, pages 355–365, 1990.

[Bar75] J. Barwise. *Admissible sets and structures.* Springer-Verlag, Berlin, 1975.

[Bar84] H.P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics.* North-Holland, Amsterdam, 1984. Second edition.

[BM92] K. Bruce and J.C. Mitchell. PER models of subtyping, recursive types and higher-order polymorphism. In *Proc. 19th ACM Symp. on Principles of Programming Languages*, pages 316–327, January 1992.

[BMM90] K. B. Bruce, A. R. Meyer, and J. C. Mitchell. The semantics of second-order lambda calculus. *Information and Computation*, 85(1):76–134, 1990. Reprinted in *Logical Foundations of Functional Programming,* ed. G. Huet, Addison-Wesley (1990) 213–273.

[BTCGS91] V. Breazu-Tannen, T. Coquand, C.A. Gunter, and A. Scedrov. Inheritance as explicit coercion. *Information and Computation*, 93(1):172–221, 1991. Reprinted in [GM94].

[Car89] F. Cardone. Relational semantics for recursive types and bounded quantification. In *ICALP*, pages 164–178, Berlin, 1989. Springer LNCS 372.

[CGW89] T. Coquand, C.A. Gunter, and G. Winskel. Domain-theoretic models of polymorphism. *Information and Computation*, 81(2):123–167, 1989.

[Cut80] N.J. Cutland. *Computability: An introduction to recursive function theory.* Cambridge Univ. Press, Cambridge, 1980.

[FRMS92] P. Freyd, G. Rosolini, P. Mulry, and D.S. Scott. Extensional PER's. *Information and Computation*, 98(2):211–227, 1992. Preliminary version appeared in *Proc. IEEE Symp. on Logic in Computer Science,* IEEE, 1990, 346–354.

[Gir86] J.-Y. Girard. The system F of variable types, fifteen years later. *Theor. Comp. Sci.*, 45(2):159–192, 1986.

[GM94] C.A. Gunter and J.C. Mitchell, editors. *Theoretical aspects of object-oriented programming.* MIT Press, Cambridge, MA, 1994.

[GS90] C.A. Gunter and D.S. Scott. Semantic domains. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B*, pages 633–674. North-Holland, Amsterdam, 1990.

[HRR90] J.M.E. Hyland, E.P. Robinson, and G. Rosolini. The discrete objects in the effective topos. *Proc. London Math. Soc.*, 60:1–36, 1990.

[Hyl82] J.M.E. Hyland. The effective topos. In *The L.E.J. Brouwer Centenary Symposium*, pages 165–216. North-Holland, Amsterdam, 1982.

[Hyl88] J.M.E. Hyland. A small complete category. *Ann. Pure and Applied Logic*, 40, 1988. Lecture delivered at the conference Church's Thesis: Fifty Years Later, Zeiss(NL), June 1986.

[Hyl90] J.M.E. Hyland. First steps in synthetic domain theory. In A. Carboni *et al.*, editor, *Category Theory, Proc. Como 1990*, pages 131–156. Springer LNM 1488, 1990.

[McC79] N. McCracken. *An Investigation of a Programming Language with a Polymorphic Type Structure.* PhD thesis, Syracuse Univ., 1979.

[Pho90a] W. Phoa. *Domain theory in realizability toposes.* PhD thesis, Cambridge, 1990. Available as University of Edinburgh Dept. of Computer Science report CST-82-91 and ECS-LFCS-91-171.

[Pho90b] W. Phoa. Effective domains and intrinsic structure. In *Proc. IEEE Symp. on Logic in Computer Science*, pages 366–377, 1990.

[Sco76] D. Scott. Data types as lattices. *Siam J. Computing*, 5(3):522–587, 1976.

[SP82] M. Smyth and G.D. Plotkin. The category-theoretic solution of recursive domain equations. *SIAM J. Computing*, 11:761–783, 1982.