PER models of subtyping, recursive types and higher-order polymorphism

Kim Bruce*
Computer Science Department
Williams College
Williamstown, MA 01267
kim@cs.williams.edu

John C. Mitchell†
Computer Science Department
Stanford University
Stanford, CA 94305
mitchell@cs.stanford.edu

Abstract

We relate standard techniques for solving recursive domain equations to previous models with types interpreted as partial equivalence relations (per's) over a D_∞ lambda model. This motivates a particular choice of type functions, which leads to an extension of such models to higher-order polymorphism. The resulting models provide natural interpretations for function spaces, records, recursively defined types, higher-order type functions, and bounded polymorphic types $\forall X <: Y.A$ where the bound may be of a higher kind. In particular, we may combine recursion and polymorphism in a way that allows the bound Y in $\forall X <: Y.A$ to be recursively defined. The model may also be used to interpret so-called "F-bounded polymorphism." Together, these features allow us to represent several forms of type and type functions that seem to arise naturally in typed object-oriented programming.

1 Introduction

In type systems aimed towards object-oriented programming, several typing ideas naturally arise. The most basic are subtyping, the fact that values of one type may be treated as values of another, polymorphism, and recursively defined types. Since many of the existing, implemented type systems for object-oriented languages have notable errors (see, *e.g.*, [Coo89]), it is important to prove the soundness of these type systems. The primary technique for doing so is by constructing a semantic model. A class of semantic models for languages with subtyping, polymorphism and recursion are developed in [Cop85, Ama91, Car89, AP90]. These models all interpret types as partial equivalence relations (per's) over suitable D_∞ structures obtained by the so-called "inverse limit" construction. The main contributions of this paper are to provide a general framework for understanding type recursion in these models and to extend per models to higher-order type functions and additional forms of polymorphism. Since the use of per's over a single domain gives us natural interpretations of subtyping and polymorphism, most of the technical effort, in the previous papers and our own, is directed toward the problem of recursively-defined types.

The first contribution of this paper is a categorical diagram relating the apparently ad hoc view of type recursion in [Cop85, Ama91, Car89] to the category of domains approximating D_∞. While similar in form to the view expressed by Abadi and Plotkin in [AP90], our "fundamental diagram" may be applied directly to the earlier models of [Cop85, Ama91, Car89]. We also use this diagram to identify a larger class of type functions with fixed points than the [AP90] construction. This allows us to interpret some useful forms of polymorphism (specifically, bounded and "F-bounded" polymorphism) that were excluded from the Abadi-Plotkin model.

Our second contribution is to extend previous models to alternate and higher-order forms of polymorphism, including solving domain equations over higher kinds. Bounded polymorphism, presented in [CW85], is extremely suggestive for object-oriented programming. The bounded polymorphic type $\forall X <: A.B(X)$ is the type of all functions which, when applied to any subtype X of A, returns an element of the type

*Part of this work was done while the author was on leave at the Computer Science Department, Stanford University, and at the DEC Systems Research Center. Partially supported by NSF grant CCR-9105316 and, at Stanford, the Powell Foundation.

†Supported in part by an NSF PYI Award, matching funds from Digital Equipment Corporation, the Powell Foundation, and Xerox Corporation; NSF grant CCR-8814921 and the Wallace F. and Lucille M. Davis Faculty Scholarship.

$B(X)$. For example, $\forall X <: int.\, X \to X$ is the type of functions which, when applied to a subtype X of *int*, maps X to X. Introduced in [CCH+89], F-bounded polymorphism arises naturally in typing a style of object-oriented programming, and appears more useful than the original form of bounded polymorphism in the presence of recursive types. The type $\forall X <: F(X).\, B(X)$ is the type of all functions which, for any type X satisfying the constraint $X <: F(X)$ have functionality given by the type $B(X)$. An alternative, discovered independently by Luca Cardelli and the second author, is to use the type $\forall G <: F.\, B(\mathit{fix}\ G)$, where G and F are both functions from types to types, in place of $\forall X <: F(X).\, B(X)$. This seems to work just as well as F-bounded quantification on all of the motivating examples, but requires higher-order polymorphism, *i.e.*, expressions parameterized by type functions and, possibly, higher-order functions from types to types.

Before discussing higher-order type recursion in our models, we give a motivating example. It might first appear that type constructors (functions from types to types) such as *list* and *tree* are recursively-defined functions. Although the usual definition,

$$list(t) = unit + t \times list(t),$$

appears to be the recursive definition of a function from types to types, we may actually de-sugar this into a definition

$$list = \lambda t{:}T.\, \mathit{fix}_T(\lambda l{:}T.\, unit + t \times l)$$

using an operator fix_T that finds the least fixed point of any function from T to T, where T is the kind (collection) of types. (Here, *unit* is the trivial type with a single element.) Since we only use a fixed point operator on type functions, as opposed to type functionals, *tree* only requires recursive definitions of types. The same is true of other familiar recursive type constructors such as *tree*.

If we try to define a type of list "objects," in the sense of object-oriented programming, then we seem to need higher-order type recursion. This example was brought to our attention by Luca Cardelli and the Abel Group at HP Labs [CCH91, Car91]. In an object-oriented program, a list object will have methods such as *empty?*, to say whether the list is empty, and *head* and *tail* to return the head and tail of the list represented by the object. The types of these methods would appear in the type of the list object, as discussed in [CM91, CHC90, CCH+89, Mit90a], for example. This gives us an ordinary recursive definition such as

$$\begin{aligned} list(t) = \{\ & head{:}\ unit \to t \\ & tail{:}\ unit \to list(t) \\ & empty?{:}\ unit \to bool\} \end{aligned}$$

in which *list* appears recursively, but the definition may be written so that only a fixed point operator for ordinary type recursion is required.

If we wish to add a *map* function to list objects, where *map* takes an function argument and applies this to every element of the list, then we would define lists by the more complicated recursive type

$$\begin{aligned} list(t) = \{\ & \ldots \\ & map{:}\forall s.(t \to s) \to list(s)\}. \end{aligned}$$

The significant aspect of the type of the *map* function is that now *list* appears with an argument other than t. For this reason, we must interpret this declaration as the result of applying a higher-order fixed-point operator to a functional that maps type functions to type functions. This is illustrated in Section 7. Another example that might be of interest is the type of list objects with a method

$$power{:}\ unit \to list(list(t)).$$

Some related kinds of polymorphism have been discussed in connection with proper extensions of ML [Myc84, Hen89, KTU89].

In this paper, we show how to extend any suitable per model to higher-order type functions by identifying a general class of "rank-ordered" sets and functions. These form a cartesian closed category such that every "rank-increasing" function has a unique fixed point. Several classes of per's over D_∞ may be rank-ordered, using ranks induced by the domains $D_0, D_1, D_2, \ldots$ approximating D_∞, allowing us to interpret type functions, type functionals, and so on as rank-preserving functions. In type-theoretic terms, this technique gives a model of F_ω with subtyping at each kind (order) and recursion operators (with a minor technical restriction on uses of recursion) at each kind. The construction may be carried out for a variety of collections of per's. In particular, we obtain one model whose types are the same as those in [Ama91, Car89] and another whose types are similar to those in [AP90]. The advantage of the latter version is that the set of elements of each type forms an ω-algebraic cpo, allowing proofs by fixed point induction.

By choosing an appropriate D_∞ model to begin with, any of the previous models could be extended to the basic form of records described in [Car88]. Although we do not discuss records at any length in this paper, it is our belief that the models given here extend to more flexible forms of record types, possibly including the type operations described in [CM91]. We hope to explore this in future work.

The models obtained by our techniques are closely related to others appearing in the literature. The paper [BL90] describes a model of subtypes and bounded polymorphism based on partial equivalence relations.

The semantic models described in [Ama91, Car89] added support for type recursion to models of subtypes and bounded polymorphism, although the latter is not discussed explicitly in [Ama91]. However, as presented, the previous models do not provide an interpretation for recursively-defined type functions or any extension of bounded polymorphism that involves type functions. In [AP90], where a similar but more structured class of models are developed, the type functions are realizable functors. However, these models do not have an obvious interpretation of bounded quantification, since this type operation is not functorial (treating $\forall x <: A.\, B$ as a function of the bound A). In this instance, the strict categorical approach does not seem adequate for treating combinations of type recursion and bounded quantification. Another class of related models are those based on partial equivalence relations over the natural numbers. In [FRMS90], a form of domain is developed within the standard effective topos over natural numbers with partial recursive function application; related work may be found in the references of that paper. The precise relation between domains in an effective setting and the effective topos over a domain does not seem clear at the time of this writing. As a final pointer into the literature, we mention that [BTCGS90, BTCGS91] describe a method for interpreting languages with subtyping and related forms of polymorphism in models of languages without subtyping.

2 Overview of the model

In concrete terms, the main idea is that we choose some suitably rich cpo D_∞, constructed as the limit of cpo's $D_0, D_1, D_2, \ldots$, and interpret types as certain partial equivalence relations (*per's*) over D_∞. Recall that a per over a set A is a symmetric and transitive binary relation on A (see [Mit86, Mit90b], for example, for further explanation). For each suitable per $R \subseteq D_\infty \times D_\infty$, there is a sequence of relations $R_{[0]}, R_{[1]}, R_{[2]}, \ldots$, determined by R and the sequence of domains $D_0, D_1, D_2, \ldots$, such that R is the limit of the $R_{[i]}$'s in a natural and entirely standard sense. We call the relation $R_{[i]}$ the *rank i approximation of* R, or sometimes the ith approximation of R for short. The type functions in the model are all functions from suitable per's to suitable per's which respect the ranked approximations of relations in a certain way. Each such function, F, has an approximating sequence, $F_{[0]}, F_{[1]}, F_{[2]}, \ldots$, and so the collection of type functions turns out to have the same abstract "rank ordered" structure as the collection of suitable pers. This allows us to repeat the definition of suitable function for higher kinds (such as functions from type functions to type functions). Since all of this is done using per's over a single domain, we have standard interpretations of subtyping, as in [BL90], and polymorphism, as in [BL90, CL90, Gir72, Mit86, Tro73]. Moreover, every function with a certain rank property has a unique fixed point. This gives us recursion operators at all kinds. Finally, all of the standard type constructors and all variations of bounded quantification mentioned in the introduction have the required rank-related properties to be included in the model.

3 PERs and the D_∞ construction

3.1 Categorical setting for recursive domain equations

We begin by describing the setting for the D_∞ construction. We work with ω-complete partial orders and two classes of maps, continuous functions and embedding-projection pairs. The reader may recall from [Bar84, GS90, SP82] that an embedding-projection pair between cpo's D and E consists of maps $e: D \to E$ and $p: E \to D$ such that $p \circ e = id_D$ and $e \circ p \leq id_E$. The map e is called the *embedding*, and p the *projection*. Each determines the other, assuming both exist, so it suffices to name the embedding or the projection. We find it convenient to work with projections and leave embeddings implicit. We write CPO for the category of ω-complete partial orders with continuous maps, and CPO^{P} for the category of ω-complete partial orders with projections.

If H is a continuous functor from CPO^{P} to CPO^{P}, then we may construct a cpo D_∞ as the inverse limit of the ω-diagram

$$\perp \xleftarrow{*} H(\perp) \xleftarrow{H(*)} H^2(\perp) \xleftarrow{H^2(*)} \cdots$$

where $\perp = \{\perp\}$ is the one-element cpo (the terminal object of both CPO and CPO^{P}) and $*$ is the unique projection from $H(\perp)$ to the terminal object. It is common notation to write D_i for $H^i(\perp)$ and say that D_∞ is the limit of the D_i's. We write p_i for the projection from D_∞ to D_i. A consequence of the continuity of H is that D_∞ is isomorphic to $H(D_\infty)$; this is in fact the definition of ω-continuity for functors. An important fact is that each D_i is isomorphic to some subdomain $\hat{D}_i \subseteq D_\infty$, with projection $\hat{p}_i$ from D_∞ to $\hat{D}_i$. If $d \in D_\infty$, we write $d_{[i]}$ for $\hat{p}_i(d)$ and note that each $d \in D_\infty$ is the limit (inside D_∞) of the ω-chain $d_{[0]} \leq d_{[1]} \leq d_{[2]} \leq \ldots$. It is worth mentioning that most functors of interest are continuous on CPO^{P}; to be continuous on CPO^{P}, it suffices to be locally continuous over CPO [SP82].

3.2 The category CPER of complete pers

A general setting for studying per's and recursion is the category CPER. The objects of this category are pairs $\langle R, D\rangle$, where D is a cpo and R is a partial equivalence relation on D which is closed under sups of ω-chains. The morphisms of CPER are pairs of continuous functions which respect the relations. More specifically, $\langle f, g\rangle$ is a morphism from $\langle R, D\rangle$ to $\langle R', D'\rangle$ if $f,\ g: D \to D'$ are continuous and $x\ R\ y$ implies $f(x)\ R\ g(y)$. This is a subcategory of the "injective scone" of CPO $\times$ CPO [AMSW91], which is a subcategory of the comma category $\langle$CPO $\downarrow$ CPO $\times$ CPO$\rangle$, also called the *scone* or *Freyd cover* of CPO $\times$ CPO [LS86]. A related category may be found in [SP82, Example 6]. The argument given in [SP82, Example 6] also shows that CPER is an **O**-category with all ω-limits. We write CPER^{P} for the subcategory of CPER with projection maps.

In the following section, we use the category CPER to motivate the definition of rank-increasing functions. Using this category, the general results of [SP82] imply that every rank-increasing function on pers has a fixed point, since each such function determines a continuous functor on CPER^{P}. A caveat, however, is that the category CPER is not the category we use to give meanings to expressions. The morphisms of CPER are continuous functions, while the meaning of a term in any of the class of models covered by this paper is an equivalence class of continuous functions. The importance of the equivalence relations are that these are required to make the models extensional.

An alternative category is the one obtained from CPER by taking equivalence classes of continuous functions as morphisms. However, there does not appear to be a natural ordering on equivalence classes unless we consider only antisymmetric pers (see Section 8 and [AP90]), and only the profinite, bifinite, or sfp domains. The arguments given in [AP90] may be used to show that this more specialized category is an **O**-category with ω-limits. The development given in the following section may be carried out for CPER, as stated, or with CPER replaced with the category whose objects are antisymmetric pers over bifinite domains and morphisms are equivalence classes of continuous functions. (Two continuous functions f, g from $R \subseteq D \times D$ to $S \subseteq E \times E$ are equivalent if $f(x)\ S\ g(y)$ whenever $x\ R\ y$.)

3.3 Continuous extensions of CPO functors to CPER

We say functor $F: \text{CPER}^{\text{P}} \to \text{CPER}^{\text{P}}$ *extends* functor $H: \text{CPO}^{\text{P}} \to \text{CPO}^{\text{P}}$ if the following conditions are satisfied:

- If $F\langle R, D\rangle = \langle R', D'\rangle$ then $D' = H(D)$.
- On morphisms, F is $H \times H$, *i.e.*, $F\langle f, g\rangle = \langle H(f), H(g)\rangle$.

We will only be interested in the case where both functors are continuous. The importance of this definition is given in the following proposition, and the associated "fundamental diagram" in Figure 1.

Proposition 3.1 *If F on CPER^{P} extends H on CPO^{P}, both continuous, then the diagram $F^n(\perp)$ in CPER^{P} is a diagram of relations over the diagram $H^n(\perp)$ in CPO^{P} and, furthermore, the limit of $F^n(\perp)$ is a relation R_∞ over the limit D_∞ of $H^n(\perp)$.*

The special case we are interested in, as suggested by the terminology, is when H is the functor on CPO which is used to construct the D_∞ model we want. Then we have "fixed points" of each continuous F extending H, obtainable as limits of a diagram consisting of relations over the D_i's. In our models, the types will be per's and we will use the fundamental diagram to solve recursive equations over these types. Note that if F extends H, the action of F on morphisms is completely determined by H. For this reason, when we fix H, we work with object maps on CPER^{P}, which are simply maps from pers to pers. This explains why we are able to give a categorical explanation to a model whose type functions are simply per maps, rather than functors on CPER.

3.4 A specific D_∞ and collection of pers

To construct a specific model, we choose a continuous functor $H: \text{CPO}^{\text{P}} \to \text{CPO}^{\text{P}}$ such that the resulting limit cpo D_∞ is rich enough to interpret the programming language expressions of interest. For concreteness, the reader may consider

$$H(D) = A + [L \to D] + [D \to D]$$

as the standard example, where A is some cpo of atomic values (say natural numbers), L is a countably infinite flat cpo of "labels" used as component names in records, and $[D \to D]$ is the cpo of all continuous functions from D to D. A technical detail that will save the careful reader some confusion when we get to recursive type definitions is that $+$ must be coalesced, as opposed to separated, sum. This is so that all types over D_∞ share the same least element. In working with bifinite domains and antisymmetric pers, A must be finite. This is not as severe a restriction as it might first appear, since a finite A here may lead to a countably infinite summand in the limit cpo, $D_\infty = H(D_\infty)$. See [AP90] for more details.

The types over D_∞ will be partial equivalence relations satisfying certain conditions.

Definition 3.2 *A per $R \subseteq D_\infty \times D_\infty$ is* nice *if*

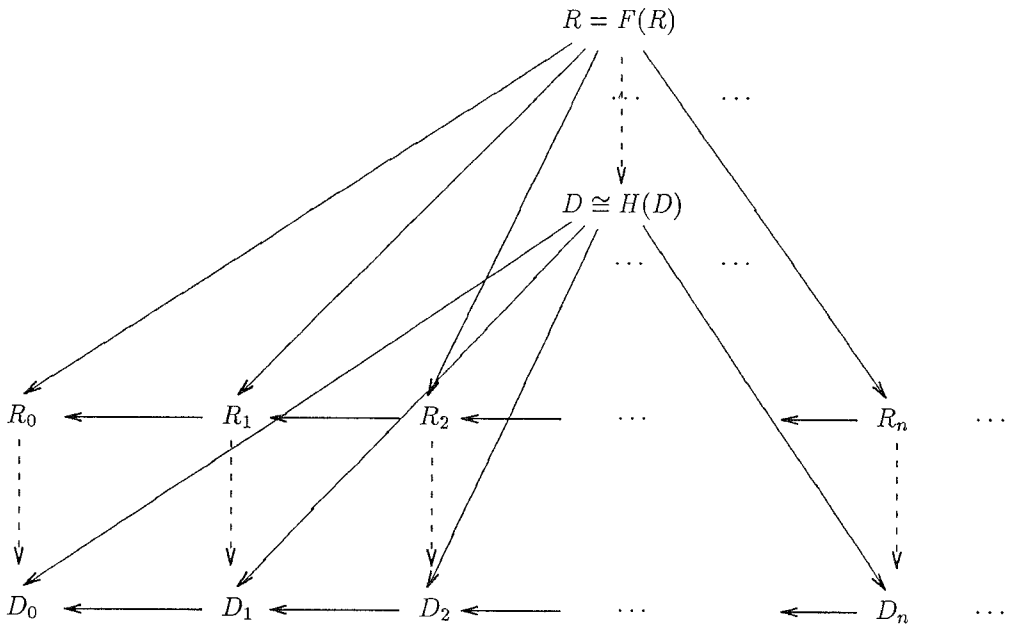

Figure 1: The Fundamental Diagram: $D_{i+1} = H(D_i)$ and $R_{i+1} = F(R_i)$.

- $\langle \bot, \bot \rangle \in R$
- *If* $\{\langle d_i, e_i \rangle \mid i > 0\}$ *is an* ω*-chain in* $D_\infty \times D_\infty$ *with limit* $\langle d, e \rangle$ *and* $\langle d_i, e_i \rangle \in R$ *for each* i*, then* $\langle d, e \rangle \in R$.
- *For all* $d, e \in D_\infty$, $\langle d, e \rangle \in R$ *iff* $\langle d_{[i]}, e_{[i]} \rangle \in R$ *for all* i.

The first two conditions are needed to give closure under limits. The third, which also appears in [Cop85, Ama91, Car89], may now be seen as the requirement that each per be the limit of a sequence of pers of the approximating domains $D_0, D_1, \ldots$ as in Figure 1. If R is a nice per then we write $R_{[i]}$ for the restriction of R to the subdomain $\hat{D}_i$ of D_∞.

Standard type constructors such as function space, records, polymorphism and bounded quantification may all be interpreted as operations on per's; see [BL90, CL90]. We give precise definitions of function space, records, and F-bounded quantification in Section 7. Ordinary "subtype bounded" quantification is covered as a special case of F-bounded quantification.

Definition 3.3 *A collection* $\mathcal{R}$ *of nice pers over* D_∞ *is acceptable if*

- $\mathcal{R}$ *contains the per,* $\{\langle \bot, \bot \rangle\}$, *and is closed under function space, records, and F-bounded quantification.*
- *If* $\{R^i \mid i < \omega\} \subseteq \mathcal{R}$ *satisfies the property that for all* $j \geq i, R^j_{[i]} = R^i$, *then there is a unique* $R \in \mathcal{R}$ *such that for all* $i, R_{[i]} = R^i$.

The first clause is clearly necessary if per's are going to interpret types in our models. (In the case that additional type constructors are required, we would add appropriate requirements to our notion of acceptable collection of per's.) The second clause on the existence of unique sups of increasing chains of per's will enable us to find unique solutions to domain equations over types. It plays the same role in our construction as the "completeness" requirement in complete partial orders.

The two specific collections of per's we mention in Section 8 are acceptable for any collection of type constructors we have considered. The first is the collection $\mathcal{R}$ of all nice pers, called CUA in [Car89] and CUPER in [Ama91]. The second is all antisymmetric pers over a D_∞ model constructed in the category of bifinite domains. For the rest of this section, we simply require some acceptable collection of per's over a suitably rich D_∞ structure. For notational simplicity, we refer to elements of this collection as *nice per's*.

3.5 A natural class of per maps

An appealing idea which is useless without further refinement is that there is a cpo structure on nice pers. If R is a nice per with $R = R_{[i]}$ for some i, then we define the *rank* of R by

$$rnk(R) = min\{i \mid R = R_{[i]}\}.$$

If $R \neq R_{[i]}$ for all i, we say $rnk(R) = \infty$. We say R *approximates* S, and write $R \lhd S$ if $rnk(R)$ is finite and

$R = S_{[rnk(R)]}$. The limit, or least upper bound, of an ω-chain $R_0 \lhd R_1 \lhd R_2 \lhd \ldots$ of nice pers with $rnk(R_i) = i$ is the relation R satisfying $d\ R\ e$ iff $d_{[i]}\ R_i\ e_{[i]}$ for all i, so the collection of nice pers becomes a cpo with each R the limit of the $R_{[i]}$'s.

Since the collection of nice pers has a cpo structure, it is plausible to consider the collection of continuous functions from nice pers to nice pers as the type functions of some model, and extend to higher kinds inductively as usual. By the standard argument showing that any continuous function on a complete partial order has a least fixed point, we would obtain recursion operators at every kind. *However, this does not work!* The most obvious problem is that the function space constructor is *not monotonic,* let alone continuous, with respect to the cpo structure on pers we have just described. In particular, if $A \lhd A'$ then $A' \to B \lhd A \to B$, rather than the reverse. This is the usual problem of the function space constructor being contravariant in its first argument. Therefore, we will have to refine the notions of cpo and continuous function in order to get a natural and suitable class of type functions.

One contribution of this paper is to identify a set of nice maps on nice pers. We say a function F from nice pers to nice pers is *rank-increasing* if for all nice per's A, and all $j \geq i-1, (F(A))_{[i]} = (F(A_{[j]}))_{[i]}$. In other words, all elements of $F(A)$ of rank i are determined by the elements of A of rank $i-1$. This definition was discovered by examining the fundamental diagram. After we had investigated rank-ordered sets with rank-preserving functions (defined below), we learned that this was a subcategory of the category of bounded ultrametrics with non-expansive maps, which has similar properties.

We relate rank properties of functions to the "fundamental diagram" as follows. Let $R \subseteq D_\infty \times D_\infty$ be a nice per. Then for each i, $\langle R_{[i]}, D_i \rangle$ is an object of CPER. If F extends H then $F\langle R_{[i]}, D_i \rangle = \langle R'_{[i+1]}, D_{i+1} \rangle$ where $F\langle R, D_\infty \rangle = \langle R', D_\infty \rangle$. If, by a slight abuse of notation, we write $R' = F(R)$, then $(F(R))_{[i]} = (F(R_{[i-1]}))_{[i]}$, as in the definition of rank-increasing maps.

Although this may be extracted from the fundamental diagram, we show directly in Section 5 that every rank-increasing function has a unique fixed point, and the collection of rank-increasing functions has the same abstract structure as an acceptable collection of nice pers. It is shown in Section 7 that the type constructors for function space, records, and F-bounded quantification are rank-increasing. This suggests that we could take the collection of type functions to be all rank-increasing functions on an acceptable collection of per's, and repeat the construction through higher kinds. The only drawback of this is that certain trivial functions, like the identity map from types to types, only satisfy a weaker *rank-preserving* property, and are not rank-increasing. Since the larger class of "rank-preserving" functions are also rank-ordered, the more natural model contains all rank-preserving functions.

A mild embarrassment we have about this construction is that only the rank-increasing subset of the rank-preserving functions have unique fixed points. A mitigating factor is that since all the type connectives are rank increasing, and the composition of rank-increasing functions with rank-preserving ones produces a rank increasing function, the only limitation that this yields is that in applying a recursion operator to a type function or functional of some order, the body of the function must involve at least one type connective or operator (such as $\to$ or F-bounded quantification). While we would like to lift this restriction on recursion, the only alternative we know of at this point is to work with the smaller class of type functions described in [AP90]. However, this would involve dropping even simple bounded quantification, which is a nontrivial trade-off. In the special case that we restrict the language to F_3, as opposed to F_ω, we are able to construct a model with unrestricted use of recursion by adding identity and projection maps to the rank-increasing functions. Since F_3 is adequate for most practical examples, this is an appealing variant of our construction.

4 The CCC of rank-ordered sets

In this section we define rank-ordered sets and show that rank-ordered sets with rank-preserving functions form a cartesian closed category (ccc). Since any acceptable collection of nice pers may be rank-ordered, this gives us an interpretation of higher-order lambda calculus in which each kind (including the collection of types, collection of type functions, etc.) is a rank-ordered set. An important fact is that every rank-increasing function on a rank-ordered set has a unique fixed point and, moreover, all of the type constructors we have considered turn out to be rank-increasing.

Definition 4.1 *A set K is* rank-ordered *if there is a map, $(\cdot)_{[i]}: K \to K$, for each $i \geq 0$, satisfying the following three conditions.*

1. *For all $A \in K, (A_{[i]})_{[j]} = A_{[min(i,j)]}$.*
2. *For all $A, B \in K$, we have $A_{[0]} = B_{[0]}$. We write Bot_K for $A_{[0]}$.*
3. *The final condition is stated using two definitions. For each i, we let $K_{[i]} = \{A_{[i]} \mid A \in K\}$. For $A, B \in K$, we write $A \lhd_i B$ if $A = B_{[i]}$. (This implies $A \in K_{[i]}$.) The final condition is that if $\{A_i\}_{i<\omega}$ is a sequence from K with $A_i \lhd_i A_{i+1}$,*

then there is a unique $A \in K$ such that for all i, $A_i = A_{[i]}$

To show that the set of rank-ordered sets forms a ccc, we must define products, morphisms and exponentials. We begin with products.

Definition 4.2 *If K and L are rank-ordered sets, we let*

$$K \times L = \{\langle k, l\rangle | k \in K, l \in L\}.$$

We rank-order $K \times L$ by defining $\langle k, l\rangle_{[i]} = \langle k_{[i]}, l_{[i]}\rangle$.

To show that $K \times L$ is a rank-ordered set, we must verify the three conditions above. The first and second follow trivially from the fact that K and L are rank-ordered. The third follows easily by choosing $\bigsqcup\langle k_i, l_i\rangle$ to be $\langle \bigsqcup_i k_i \bigsqcup_i l_i\rangle$, if $\{\langle k_i, l_i\rangle\}_{i\leq\omega}$ is an increasing chain.

The morphisms between rank-ordered sets are functions which are defined smoothly with respect to approximations to terms of restricted ranks.

Definition 4.3 *A function $F: K \to L$ is* rank-preserving *if for all $j \geq i, (F(A))_{[i]} = (F(A_{[j]}))_{[i]}$ and* rank-increasing *if for all $j \geq i-1, (F(A))_{[i]} = (F(A_{[j]}))_{[i]}$. We write $[K \Rightarrow L]$ for the collection of rank-preserving functions and $[K \stackrel{i}{\Rightarrow} L]$ for the collection of rank-increasing functions. We rank-order $[K \Rightarrow L]$ by defining $F_{[i]} = \lambda A \in K.(F(A))_{[i]}$.*

It is straightforward to verify that the collection of rank-preserving functions from K to L is a rank-ordered set. In addition, since the limit of a sequence of rank-increasing functions is rank-increasing, the collection of rank-increasing functions from K to L is a rank-ordered set.

Theorem 4.4 *The collection of rank-ordered sets, with rank-preserving functions, is a Cartesian closed category.*

5 Properties of rank-increasing functions

In this section we identify some useful closure properties of rank-increasing functions. In particular, the fixed point of a rank-increasing function is rank-increasing, and the composition of a rank-increasing function with a rank-preserving (or increasing) one is rank-increasing. For the rest of this section, we assume that K, L, and M are rank-ordered sets. To ease readability, we also write $F(A, B)$ rather than $F(\langle A, B\rangle)$.

Lemma 5.1 *Let $F: K \times L \to M$. Then F is rank-preserving (respectively, rank-increasing) iff it is rank-preserving (respectively, rank-increasing) in each of its arguments separately. In particular,*

1. *If F is rank-preserving, then*

$$(F(A, B))_{[i]} = F(A_{[i]}, B)_{[i]} = (F(A, B_{[i]}))_{[i]}.$$

2. *If F is rank-increasing, then*

$$(F(A, B))_{[i]} = F(A_{[i-1]}, B)_{[i]} = (F(A, B_{[i-1]}))_{[i]}.$$

The advantage of rank-increasing functions is that each endo-function has a unique fixed point. In addition, the fixed point of a multi-argument function that is rank-increasing in one argument retains its rank-related properties in other arguments.

Theorem 5.2 *If $F: K \times L \to K$ is rank-increasing in its first argument and rank-preserving in its second, then there is a unique function, $G: L \to K = \mu t.F(t, S)$, returning the fixed point of F in its first argument. Moreover, if F is rank-increasing in its second argument, then G is rank-increasing.*

Proof. Sketch: Construct $G(S)$ such that $G(S) = F(G(S), S)$ by constructing an approximating chain for G. Define $G_0(S) = \mathrm{Bot}_K$, and, for $i \geq 1$, $G_i(S) = (F(G_{i-1}(S), S))_{[i]}$. One can show by induction that $G_i(S) \lhd_i G_{i+1}(S)$, for all i.

Let $G(S) = \bigsqcup_i G_i(S)$. Then

$$\begin{aligned} F(G(S), S) &= \bigsqcup_i (F(G(S), S))_{[i]} \\ &= \bigsqcup_i (F((G(S))_{[i-1]}, S))_{[i]} \\ &= \bigsqcup_i G_i(S) \\ &= G(S). \end{aligned}$$

It is not difficult to show that G is unique and rank-preserving (rank-increasing), if F is rank-preserving (respectively, rank-increasing) in its second argument. ∎

Lemma 5.3 *Let $G: K \to L$, and $F: L \to M$. If at least one of F and G is rank-increasing and the other is rank-preserving, then $F \circ G$ is rank-increasing.*

6 Partial equivalence relations over D_∞

In this section we present the fundamental notions involved in the construction of specialized partial equivalence relations over D_∞ models of the untyped lambda calculus. As in Section 3, we assume that D_∞ is constructed as the inverse limit of a continuous functor $H: \mathrm{CPO}^{\mathrm{P}} \to \mathrm{CPO}^{\mathrm{P}}$. For example, let A be a fixed CPO representing "atomic" types, L a flat domain corresponding to a set of "labels", and "+" be the coalesced sum over cpo's. Then a model constructed from the functor whose definition on objects is $H(D) = A + [L \to D] + [D \to D]$, results in a cpo, D_∞, such that

$$D_\infty \cong A + [L \to D_\infty] + [D_\infty \to D_\infty].$$

The following definition is taken from [Car89].

Definition 6.1 *A* notion of approximation over a cpo D *is a family of continuous mappings* $(\cdot)_{[n]}: D \to D$ *satisfying the following conditions for all* $d \in D$*:*

1. $d_{[0]} = \bot$.
2. $(d_{[n]})_{[m]} = (d_{[m]})_{[n]} = d_{[min(m,n)]}$.
3. $d = \bigsqcup_i d_{[i]}$.
4. *If* $a \in L \cup A$, *then for all* $0 < i < \omega, a_{[i]} = a$.
5. *If* $f \in [D \to D]$ *and* $n \leq k$, *then* $f_{[n+1]}(d_{[k]}) = f_{[n+1]}(d_{[n]})$.
6. *If* $f \in [D \to D]$ *and* $n \leq k$, *then* $(f_{[k+1]}(d_{[n]}))_{[n]} = f_{[n+1]}(d_{[n]})$.
7. *If* $f \in [D \to D]$ *then* $f_{[n+1]}(d_{[n]}) = f_{[n+1]}(d) = (f(d_{[n]}))_{[n]}$.
8. $f = f_{[n+1]}$ *if and only if for all* $d \in D, f(d) = (f(d_{[n]}))_{[n]}$.

Since these properties were extracted from the D_∞ model construction. The following proposition should not come as a surprise.

Proposition 6.2 *There exists a notion of approximation over* D_∞.

The following properties will be useful in several technical proofs later on.

Proposition 6.3 *Every notion of approximation over D satisfies the following:*

1. *If* $n \leq m$ *then* $d_{[n]} \sqsubset d_{[m]}$, *where* $\sqsubset$ *is the ordering in the cpo D.*
2. *If* $d \in [D \to D]$ *and* $e \in D$ *then* $d(e) = \bigsqcup_i d_{[i+1]}(e_{[i]})$.

Recall the definition of nice pers and acceptable collections from Section 3. We define a notion of rank on nice per's as follows:

Definition 6.4 *If* R *is a nice per, let* $R_{[n]} = \{\langle d_{[n]}, e_{[n]}\rangle | \langle d, e\rangle \in R\} = R \cap (D_n \times D_n)$. *(Note* $R_{[n]}$ *is nice.) Write* $R \lhd_n R'$ *iff* $R = R'_{[n]}$.

Note that if R is nice, and $R \lhd_n R'$, then $\langle d, e\rangle \in R'$ implies $\langle d_{[n]}, e_{[n]}\rangle \in R$. Also, note that for all n, $R_{[n]} \lhd_n R$ and $R_{[n]} \lhd_n R_{[n+1]}$.

Proposition 6.5 *If* $\mathcal{R}$ *is an* acceptable *collection of nice per's then* $\mathcal{R}$ *is a rank-ordered set.*

Proof. Since each $R \in \mathcal{R}$ is nice, elements in pairs from R satisfy all of the properties in Definition 6.1. Thus point 1 of the definition of rank-ordered sets (Definition 4.1) follows from point 2 of Definition 6.1. Point 2 of rank-ordering is satisfied by $\{\langle \bot, \bot\rangle\}$. The last point in the definition of rank-ordering is the last condition in the definition of an acceptable collection of per's (Definition 3.3). ∎

7 Models of F_ω with F-bounded quantification

In this section we describe models of F_ω with F-bounded quantification whose types are elements of an acceptable collection of per's. We use the terminology of [BMM90]. In particular, a model involves a set $\mathcal{K}$ of *kinds*, which includes the collection T of all types as well as the product $K_1 \times K_2$ and function kind $K_1 \Rightarrow K_2$ for any $K_1, K_2 \in \mathcal{K}$. For the rest of this section, let $\mathcal{R}$ be a fixed, acceptable set of per's.

Definition 7.1 *The* kind structure, $\mathcal{K}$, *generated from* $\mathcal{R}$ *is the smallest collection of rank-ordered sets which contains* $\mathcal{R}$ *and is closed under* $\Rightarrow$ *and* $\times$, *where these operations are as defined in Section 4.*

It follows from our previous results that a kind structure is a ccc. As in [BMM90], we will use the kind structure over $\mathcal{R}$ to provide the interpretation of types, as well as all the higher kinds of a model of F_ω with F-bounded quantification. We will interpret the set of all types as the set $\mathcal{R}$. We already know by the definition of an acceptable collection of per's, that $\mathcal{R}$ is closed under our type operators: function space, records, and F-bounded quantification.

We must show that the type operators themselves are elements of the appropriate kinds. Before proving this, we give precise definitions of each type operator. In order to define higher-order F-bounded quantification, we order the elements of each kind. For types, the relevant order is the subtype ordering; for type functions, we use the induced pointwise order, as in the following definition.

Definition 7.2 *Let* $\mathcal{K}$ *be the kind structure generated from* $\mathcal{R}$. *If* $A, B \in \mathcal{R}$, *define* $A \leq_{\mathcal{R}} B$ *iff* $A \subseteq B$. *Suppose* $\leq_K$ *and* $\leq_L$ *have been defined. For* $F, G \in [K \Rightarrow L]$, *define* $F \leq_{[K \Rightarrow L]} G$ *iff for all* $A \in K$, $F(A) \leq_L G(A)$. *For* $\langle A, B\rangle, \langle C, D\rangle \in [K \times L]$, *define* $\langle A, B\rangle \leq_{[K \times L]} \langle C, D\rangle$ *iff* $A \leq_K C$ *and* $B \leq_L D$.

The following lemma is easily proved by induction on the construction of kinds.

Lemma 7.3 *Let* $K \in \mathcal{K}$ *with* $A, B \in K$. *Then*

1. $A_{[i]} \leq_K A_{[i+1]} \leq_K A$.
2. *If* $A \leq_K B$ *then* $A_{[i]} \leq_K B_{[i]}$.

We now define the type operators. Note that our definition of F-bounded quantification follows the form used in [BL90] for bounded quantification. It differs from the definition using a simple intersection which is more commonly used (see [Mit86] for the more traditional definition). Our slightly more complex definition is necessary to make the function rank-increasing.

Definition 7.4 *1. If $A, B \in \mathcal{R}$, let*
$A \to B = \{\langle d, e\rangle \in D_\infty \times D_\infty \mid$ *for all* $\langle a, b\rangle \in A, \langle d \cdot a, e \cdot b\rangle \in B\}$.

2. If $\{l_i, \ldots, l_n\} \subseteq L$ and $A_1, \ldots, A_n \subseteq \mathcal{R}$, then let $\{l_1 : A_1, \ldots, l_n : A_n\} = \{\langle d, e\rangle \in D_\infty \times D_\infty \mid$ *for all* $l_i, \langle d \cdot l_i, e \cdot l_i\rangle \in A_i\}$.

3. Let $K \in \mathcal{K}$. If $A \in [K \Rightarrow K]$ and $G \in [K \Rightarrow T]$ then let
$\forall s <: A(s).G(s) = \{\langle d, e\rangle \in D_\infty \times D_\infty \mid$ *for all* $a, b \in D_\infty$, *for all* $R \leq_K A(R)$ *such that* $R \in K$, $\langle d \cdot a, e \cdot b\rangle \in G(R)\}$.

4. If $F \in [K \Rightarrow K]$ then let
$\mu s.F(s) =$ *the least* $A \in K$ *such that* $F(A) = A$.

Note that simple bounded quantification can be obtained from F-bounded by choosing the type bound to be a constant function.

The following theorem shows that our model is closed under the above constructions. Recall that $K_1 \stackrel{i}{\Rightarrow} K_2$ is the set of rank-increasing functions from K_1 to K_2, which is a subset of the set of rank-preserving functions from K_1 to K_2.

Theorem 7.5 *Let $\mathcal{K}$ be the kind structure generated by $\mathcal{R}$, and let $K, L,$ and $M \in \mathcal{K}$. Then*

1. $H(X, Y) = X \to Y \in [T \times T \stackrel{i}{\Rightarrow} T]$.

2. Let $\{l_i, \ldots, l_n\} \subseteq L$. Then $H(X_1, \ldots, X_n) = \{l_1 : X_1, \ldots, l_n : X_n\} \in [T^n \stackrel{i}{\Rightarrow} T]$.

3. If $A \in [K \Rightarrow K]$ and $G \in [K \Rightarrow T]$ then $H(A, G) = \forall s <: A(s).G(s) \in [[K \Rightarrow K] \times [K \Rightarrow T] \stackrel{i}{\Rightarrow} T]$.

4. If $F \in [K \times K' \stackrel{i}{\Rightarrow} K]$, then $H(S) = \mu t.F(t, S) \in [K' \stackrel{i}{\Rightarrow} K]$. Moreover if $F \in [K \times K' \Rightarrow K]$ is rank-increasing in its first argument, then $H(S) = \mu t.F(t, S) \in [K' \Rightarrow K]$.

Proof. We just include the proofs for the last two cases in this conference paper.

3. Let $H(A, G) = \forall s <: A(s).G(s)$. Show $(H(A_{[i-1]}, G_{[i-1]}))_{[i]} = (H(A, G))_{[i]}$.

Let $\langle d, e\rangle \in (H(A_{[i-1]}, G_{[i-1]}))_{[i]}$ and $R \leq_K A(R)$. Thus $R_{[i-1]} \leq_K (A(R))_{[i-1]} = A_{[i-1]}(R_{[i-1]})$ by Lemma 7.3 and the definition of rank-preserving function. As a result, for all $a, b \in D_\infty, \langle d \cdot a, e \cdot b\rangle \in G_{[i-1]}(R_{[i-1]}) = (G(R_{[i-1]}))_{[i-1]} = (G(R))_{[i-1]} \subseteq G(R)$, since G is rank-preserving. Hence $\langle d, e\rangle \in (H(A, G))_{[i]}$.

Let $\langle d, e\rangle \in (H(A, G))_{[i]}$. Therefore for all $a, b \in D_\infty$ and all $R \leq_K A(R), \langle d \cdot a, e \cdot b\rangle \in G(R)$. Let $R \leq_K A_{[i-1]}(R) \leq_K A(R)$ and $a, b \in D_\infty$. Therefore $\langle d \cdot a, e \cdot b\rangle = \langle (d \cdot a)_{[i-1]}, (e \cdot b)_{[i-1]}\rangle \in (G(R))_{[i-1]} = G_{[i-1]}(R)$. Hence $\langle d, e\rangle \in (H(A_{[i-1]}, G_{[i-1]}))_{[i]}$.

Thus $H \in [[K \Rightarrow K] \times [K \Rightarrow T] \stackrel{i}{\Rightarrow} T]$

4. Let $F(T, S) \in [K \times K' \stackrel{i}{\Rightarrow} K]$. We already showed in Lemma 5.2 that $H(S) = \mu t.F(t, S)$ is rank-increasing, and hence is in $[K' \stackrel{i}{\Rightarrow} K]$. Lemma 5.2 also gives the result for F rank-increasing in its first argument and rank-preserving in its second. ∎

Thus we may use the definitions given above of types and functions from types to types in order to construct a model of the ω-order typed lambda calculus, F_ω, with F-bounded quantification.

Theorem 7.6 *Let $\mathcal{K}$ be the kind structure generated by $\mathcal{R}$. If we interpret T, the set of types, as $\mathcal{R}$, and the higher kinds as the appropriate elements of $\mathcal{K}$, then we obtain a model for the ω-order typed lambda calculus, F_ω, with subtyping, F-bounded quantification, and records. This model has the property that every rank-increasing function on any kind has a unique fixed point.*

The proof combines arguments in [BL90] for subtyping with the fixed-point properties of rank-increasing functions developed here, and direct verification of conditions for F-bounded quantification. Details are omitted from this conference paper.

Returning to the example presented in the introduction, we illustrate how to define $list : T \Rightarrow T$ such that

$$list(t) = \{\ head : unit \to t \\ tail : unit \to list(t) \\ empty? : unit \to bool \\ map : \forall s.(t \to s) \to list(s)\}$$

where *unit* represents a fixed type with one element and *bool* is a fixed type with two (non-$\perp$) elements. Define $F : (T \Rightarrow T) \Rightarrow T \Rightarrow T$ by

$$F(\ell)(t) = \{\ head : unit \to t \\ tail : unit \to \ell(t) \\ empty? : unit \to bool \\ map : \forall s.(t \to s) \to \ell(s)\}$$

Because the record constructor is rank-increasing, F is rank-increasing. Thus F has a least fixed point, *list*. Other higher-order domain equations are solved similarly.

8 Acceptable collections of per's

In this section we show that two natural collections of per's are acceptable, thus giving us two models of F_ω with F-bounded quantification. We begin by working with CUA, the collection of all nice per's, and then work with a restriction of CUA which ensures that the set of elements of any type forms a cpo.

Theorem 8.1 CUA *is an acceptable collection of* per*'s.*

Proof. We only prove closure under F-bounded quantification in this conference paper.

Let $\mathcal{K}$ be the kind structure generated from CUA and let $K \in \mathcal{K}$. If $A \in [K \Rightarrow K]$ and $G \in [K \Rightarrow \text{CUA}]$ then recall $\forall s <: A(s).G(s) =_{def} \{\langle d, e\rangle \in D_\infty \times D_\infty |$ for all $\langle a, b\rangle \in D_\infty$, for all $R \leq_K A(R)$, $\langle d \cdot a, e \cdot b\rangle \in G(R)\}$. We must show that $B =_{def} \forall s <: A(s).G(s) \in \text{CUA}$.

Clearly, B is a per and $\langle \bot, \bot\rangle \in B$. We must show that it is closed under sup's of increasing chains and approximations.

Suppose $\langle d^i\rangle$ and $\langle e^i\rangle$ are increasing chains in D such that for all $i \leq \omega$, $\langle d^i, e^i\rangle \in B$. Let $d = \bigsqcup_i d^i$ and $e = \bigsqcup_i e^i$ We must show $\langle d, e\rangle \in B$. That is, we must show that for all $R \leq_K A(R)$, and all $a, b \in D_\infty$, that $\langle d \cdot a, e \cdot b\rangle \in G(R)$. We already know that $\langle d^i \cdot a, e^i \cdot b\rangle \in G(R)$ and $G(R) \in \text{CUA}$. Thus $\langle d \cdot a, e \cdot b\rangle = \langle \bigsqcup_i (d^i \cdot a), \bigsqcup_i (e^i \cdot b)\rangle = \bigsqcup_i \langle d^i \cdot a, e^i \cdot b\rangle \in G(R)$, where the first equality holds since "$\cdot$" is continuous. Thus $\langle d, e\rangle \in B$.

The proof of closure under approximations is similar. ∎

Thus we can use CUA as the basis for our first model of F_ω with F-bounded quantification. This model is an extension of those in [Ama91] and [Car89].

Our next model is based on a construction in [AP90].

Lemma 8.2 *(Abadi-Plotkin) For any appropriately chosen continuous functor H over bifinite cpo's, there is an intrinsic preorder, $\leq_S$, for each nice per, S, over the corresponding limit, $D_\infty = H(D_\infty)$, such that*

1. *$\leq_S$ is the least complete preorder containing $\sqsubseteq$ and S.*
2. *$f \leq_{S \to T} g$ implies that for all $x \in |S|$, $f(x) \leq_T g(x)$. (Similarly for $\times$ and F-bounded quantification.)*
3. *$a \leq_S b$ implies that for all $i \leq \omega$, $a_{[i]} \leq_{S_{[i]}} b_{[i]}$.*

An additional condition is needed to partially order equivalence classes.

Definition 8.3 *We say that a* per*, S, is* antisymmetric *if for all $x, y \in S$, if $x \leq_S y \leq_S x$ then $\langle x, y\rangle \in S$. Define* ACUA $= \{S \in \text{CUA} | S$ *is antisymmetric*$\}$.

The important fact about ACUA is that the set of equivalence classes of each per in ACUA forms a cpo. Let $|S| = \{x \in D | \langle x, x\rangle \in S\}$ and $Q(S) = \{[x]_S | x \in |S|\}$. For $x, y \in |S|$, define $[x]_S \leq [y]_S$ iff $x \leq_S y$. Antisymmetry ensures that this is well-defined.

Theorem 8.4 *(Abadi-Plotkin) If $S \in$* ACUA*, then $\langle Q(S), \leq\rangle$ is an ω-algebraic cpo.*

Theorem 8.5 ACUA *is an acceptable collection of per's.*

We can add different conditions to CUA's to get other models as long as the resulting collection of per's is "acceptable." For instance, [AP90], defines the collection of "good" per's to be CUA's which also satisfy "meet-closure" and "convexity" conditions. This set also forms an acceptable collection of per's. For any such condition, we obtain a model with the properties stated in Theorem 7.6.

9 Summary

In this paper, we have described a general technique for constructing per models of F_ω (higher-order lambda calculus) that support function spaces, records, and higher-ordered F-bounded polymorphism, and that also contain solutions to all non-trivial recursive domain equations. "Acceptable" collections of "nice" per's, built over a model of D_∞, serve as the collection of types of the model. Key properties in the definition of "acceptable" collections of per's are that they are closed under the type operations of the model, and they satisfy a closure condition relating to sup's of "approximation" chains of per's. The approximation ordering is derived from the ranks of elements in the D_∞ construction. The closure condition on "approximation" chains enables the solution of recursive domain equations.

Since the construction is general, different models can be built from different acceptable collections of per's. The first model considered extended that presented (independently) by Amadio [Ama91] and Cardone [Car89] by adding functions of higher kinds and allowing F-bounded quantification over elements of higher kinds, while still providing solutions to all non-trivial (higher-order) domain equations.

The types in the second model are similar to those in the model of Abadi and Plotkin [AP90], but the set of functions from kinds to kinds is larger (we do not restrict to realizable functors) than that given in their model. A major contribution of the Abadi-Plotkin model is that all types are cpo's, presenting the possibility of reasoning by fixed point induction. By keeping their type structure, but allowing a larger set of functions from kinds to kinds, we were able to solve domain equations involving both simple and F-bounded quantification, something which does not seem to be possible in Abadi and Plotkin's original model.

A side issue that we have not explored is the connection between an F-bounded type $\forall X <: F(X).\, B(X)$ and the type $\forall G <: F.\, B(fix\ G)$ defined using a fixed-point operator in place of F-bounded quantification. Since both type expressions make sense in our model, provided we interpret "$\forall G <: F$" as quantifying over

rank-increasing functions, and both seem to serve the same purpose in practical examples, it would be interesting to explore their semantic connections.

Acknowledgements: Thanks to Gordon Plotkin, Peter Freyd and Martin Abadi for comparative comments and explanation of their related work, and to Luca Cardelli and other participants in the 1991 Stanford "subtypes seminar."

References

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

[AMSW91] S. Abramsky, J. Mitchell, A. Scedrov, and P. Wadler. Relators. Manuscript, 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.

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

[BL90] K. Bruce and G. Longo. A modest model of records, inheritance and bounded quantification. *Information and Computation*, 87(1/2):196–240, 1990.

[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.

[BTCGS90] V. Breazu-Tannen, T. Coquand, C.A. Gunter, and A. Scedrov. Computing with coercions. In *Proc. ACM Conference on Lisp and Functional Programming*, pages 44–61, 1990.

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

[Car88] L. Cardelli. A semantics of multiple inheritance. *Information and Computation*, 76:138–164, 1988. Special issue devoted to *Symp. on Semantics of Data Types*, Sophia-Antipolis (France), 1984.

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

[Car91] L. Cardelli. Examples of type recursion. Personal communication, 1991.

[CCH+89] P. Canning, W. Cook, W. Hill, J. Mitchell, and W. Olthoff. F-bounded quantification for object-oriented programming. In *Functional Prog. and Computer Architecture*, pages 273–280, 1989.

[CCH91] P. Canning, W. Cook, and W. Hill. Examples of type recursion. Personal communication, 1991.

[CHC90] W. Cook, W. Hill, and P. Canning. Inheritance is not subtyping. In *Proc. 17th ACM Symp. on Principles of Programming Languages*, pages 125–135, January 1990.

[CL90] L. Cardelli and G. Longo. A semantic basis for Quest. Technical Report 55, DEC Systems Research Center, 1990. To appear in *J. Functional Programming.*

[CM91] L. Cardelli and J.C. Mitchell. Operations on records. *Math. Structures in Computer Science*, 1(1):3–48, 1991. Summary in *Math. Foundations of Prog. Lang. Semantics*, Springer LNCS 442, 1990, pp 22–52.

[Coo89] W.R. Cook. A proposal for making Eiffel type-safe. In *European Conf. on Object-Oriented Programming*, pages 57–72, 1989.

[Cop85] M. Coppo. A completeness theorem for recursively-defined types. In *Proc. ICALP*, pages 120–130, Berlin, 1985. Springer LNCS 194.

[CW85] L. Cardelli and P. Wegner. On understanding types, data abstraction, and polymorphism. *Computing Surveys*, 17(4):471–522, 1985.

[FRMS90] P. Freyd, G. Rosolini, P. Mulry, and D.S. Scott. Extensional PER's. In *Proc. IEEE Symp. on Logic in Computer Science*, pages 346–354. IEEE, 1990.

[Gir72] J.-Y. Girard. Interpretation fonctionelle et elimination des coupures de l'arithmetique d'ordre superieur. These D'Etat, Universite Paris VII, 1972.

[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.

[Hen89] F. Henglein. *Polymorphic Type Inference and Semi-Unification.* PhD thesis, Rutgers University, April 1989. also NYU Technical Report 443, May 1989.

[KTU89] A. Kfoury, J. Tiuryn, and P. Urzyczyn. Computational consequences and partial solutions of a generalized unification problem. In *Proc. IEEE Symp. on Logic in Computer Science*, June 1989.

[LS86] J. Lambek and P.J. Scott. *Introduction to Higher-Order Categorical Logic.* Cambridge University Press, Cambridge, U.K, 1986.

[Mit86] J.C. Mitchell. A type-inference approach to reduction properties and semantics of polymorphic expressions. In *ACM Conference on LISP and Functional Programming*, pages 308–319, August 1986. Revised version in *Logical Foundations of Functional Programming,* ed. G. Huet, Addison-Wesley (1990) 195–212.

[Mit90a] J.C. Mitchell. Toward a typed foundation for method specialization and inheritance. In *Proc. 17th ACM Symp. on Principles of Programming Languages*, pages 109–124, January 1990.

[Mit90b] J.C. Mitchell. Type systems for programming languages. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B*, pages 365–458. North-Holland, Amsterdam, 1990.

[Myc84] A. Mycroft. Polymorphic type schemes and recursive definitions. In *Proc. 6th Int. Conf. on Programming, LNCS 167*, 1984.

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

[Tro73] A.S. Troelstra. *Mathematical Investigation of Intuitionistic Arithmetic and Analysis.* Springer LNM 344, Berlin, 1973.