1982

A finitary retract model for the polymorphic lambda-calculus

Nancy McCracken
Syracuse University, njmccrac@syr.edu

83-2

A finitary retract model for the polymorphic lambda-calculus

Nancy Jean McCracken

SCHOOL OF COMPUTER
AND INFORMATION SCIENCE
SYRACUSE UNIVERSITY

This paper is a preprint of a paper submitted for publication to Information and Control.

A finitary retract model for the polymorphic lambda-calculus

Nancy Jean McCracken

Syracuse University

1982

This work was partially supported by National Science Foundation grant MCS80-04219.

I. Introduction

There has been great interest in recent years in designing programming languages which permit functions which may accept types as parameters (*polymorphic functions*) and types with type parameters (*type generators*). Unfortunately, the semantics of such functions and types has not been as well understood as their practical use in software design. In [McCracken 1979], a denotational semantics was given for a simple programming language with these features. The semantics used closures, which are a special case of the more familiar retractions, over the Scott universal domain, $P\omega$, to represent types. It was then possible to interpret polymorphic functions as continuous functions from types to objects in $P\omega$ and the type generators as continuous functions from types to types. However, the model depends heavily on the fact that $P\omega$ and the types over it are all complete lattices, while in many cases it seems that the less restrictive complete partial orders are more natural for programming language semantics.

An attempt to construct a model using retracts over complete lattices was made in [Donahue 1979], but it has been shown that the construction is not valid [McCracken 1980]. Structures that may solve this problem have been suggested by Scott [Scott 1980]. In this paper, we use Scott's suggestion of finitary retracts over a finitary complete partial order. We show that Scott's conjecture that these structures will provide a model for the polymorphic functions is true. In addition, we show that this also provides a model for the type generators and that recursive types can be interpreted in the model.

In this paper, we first define a prototype language as a typed λ-calculus extended with polymorphic functions and type generators. Next, we present the finitary cpo's and the various functions and operators necessary in the semantics. And finally, we show how to use the cpo's to give a model for the language.

We are using type generators here for essentially the same construct that has also been called parameterized types or generic types. We prefer the term type generators because it emphasizes both the functional nature of our semantics and the generality of the construct. In particular, we wish to emphasize that a type generator is not a type itself. In our semantics, a type generator will have a denotation as a function from types to types, in contrast to the algebraic approach [ADJ 1979], where a parameterized data type (in their terminology) is a specification "scheme" or a class of specifications.

II. The Prototype Language

Syntactically, polymorphic functions and type generators can be introduced into a programming language by including type variables, which are distinct from ordinary variables, and by allowing abstraction with respect to type variables both in the programming language expressions (to give polymorphic functions) and in the type expressions (to give type generators). Since our goal is to have type abstraction be as general a mechanism as possible, the question naturally arises "Can type abstractions be applied to any type expression whatsoever?". For polymorphic functions the answer is essentially "yes"; if the type abstraction is formed with respect to a type variable that avoids certain special bound variable clashes, then a polymorphic function can be applied to any type expression.

For parameterized types, though, the answer is "no". Since we are representing them as functions from types to types, unrestricted type abstraction and application would make the type language into an untyped λ-calculus. This is undesirable since we want every programming language expression to have a type expression which can be interpreted as a "base" type and does not represent some unlimited sequence of type computations. The solution is to introduce a type structure for the type language, i.e. it itself becomes a typed λ-calculus. We will call the types of types "kinds" to distinguish the two levels of type structure.

The prototype language, then, is actually a hierarchy of three languages, where the programming language has for type structure a language of type expressions, which itself has a type structure of kinds. As we define the expressions in these languages, their type structure, and computation rules, we will assume that the reader is familiar with many of the standard definitions and terminology from the traditional λ-calculus.

The kinds of type expressions, Kexp

We make the simplifying assumption here that there is just one collection of types, the base types B, which can be used as types of programming language expressions, i.e. only type generators are not base types. This is certainly sufficient for our simple applicative language - for a more extensive analysis of the different roles of types in a programming language, it may be necessary to subdivide the base types (see e.g. Reynolds' use of "data types" and "program types" in [Reynolds 1978]). Consequently, the kinds of type expressions only classify the type expressions as to their functionality, i.e. whether they are a type generator or not.

Definition : Kexp is the least set satisfying

i. $B \in$ Kexp

ii. if $k,m \in$ Kexp, then $k \Rightarrow m \in$ Kexp

We will use k and m to denote arbitrary elements of Kexp.

The type expressions, Texp

The language of type expressions will provide the type structure for the programming language and is also a typed λ-calculus itself. The typed λ-expressions are to be interpreted as type generators and will have the usual β-reduction to generate a type, given a type parameter. Several other type constructors are introduced.

Let TV be a countably infinite set of type variables: s,t,...

Let TC be a set of type constants (which may include things like Int and Bool).

Let a type assignment TQ be a function from a finite subset of TV to Kexp. Then we define Texp to be a family of minimal sets of expressions for every type assignment TQ and every k in Kexp. We use the notation Texp[TQ,k] to denote the set of type expressions whose kind is k with respect to the type assignment TQ: In this definition, let $t\in TV$, $tc\in TC$, $a\in Texp[TQ,k]$, $b\in Texp[TQ,m]$:

	other conditions	expression	$\in Texp[TQ, \_\_\_]$
type constants:		tc	B
type variables:		t	TQ(t)
types of functions:		$a\to b$	B
types of products:		$a\times b$	B
types of unions:		a+b	B
recursive types:		$\mu t.a$	B
types of polymorphic fns:		$\Delta t{:}k.a$	B
type generators:			
	$\forall b\in Texp[(TQ\mid t{:}k),m]$	$\lambda t{:}k.b$	k=>m
	$\forall g\in Texp[TQ,k{=>}m]$	g[a]	m

Note: (TQ|t:k) is the type assignment $\lambda s\in TV$. <u>if</u> s=t <u>then</u> k <u>else</u> TQ(s).

As in a typed λ-calculus, we can define substitution and use it to define the computation rules of the language, α and β -conversions. For a full description of substitution and other syntactic details, see [McCracken 1979]. The language that appears there differs only by syntactic abbreviations for recursive types and the types of polymorphic functions.

> We use {b/t}a to denote the substitution of a type expression b in Texp[TQ,m] for all free occurrences of the type variable t in the type expression a in Texp[(TQ|t:m),k]. We can show that {b/t}a is in Texp[TQ,k].

Definition: Computation (Reduction) rules

(α) $\lambda t{:}k.a \;\triangleright\; \lambda s{:}k.\,\{s/t\}a$ where $s \notin \mathrm{dom}(TQ) \cup \{t\}$

(β) $(\lambda t{:}k.a)[b] \;\triangleright\; \{b/t\}a$

Conversion, denoted a cnv b, is the least substitutive equivalence relation that contains the computation rules. If a cnv b and a is in Texp[TQ,k], then b is in Texp[TQ,k]. Since Texp is a typed λ-calculus, every type expression has a normal form unique up to α-conversion.

The programming language expressions, Plexp

(or polymorphic λ-calculus expressions)

The programming language Plexp is an extension of the ordinary typed λ-calculus, where λ-abstraction and application represent functions with ordinary arguments. To represent polymorphic functions with type arguments, we add another abstraction operator, Λ , which binds type variables. The addition of type abstraction and application does not completely parallel ordinary abstraction and application in this language since type expressions cannot occur directly as objects in the language, i.e. they are not "first-class" objects. The role of type expressions is to indicate the types of ordinary objects in the language and the effect of type application is to compute the types of objects.

Let V be a countably infinite set of variables x,y,z...

Let C be a set of constants with type expressions assigned by a function QC, where $\forall c \in C$, QC(c) is in Texp[$\emptyset$,B].

Let an ordinary assignment Q be a function from a finite subset of V to Texp.

In the following definition of well-typed expressions, it is necessary that the types in the image of Q be themselves correctly typed according to a particular type assignment TQ. We will call this property the "compatibility of Q with TQ" and define it formally:

$$\mathrm{compat}(Q,TQ) \text{ iff } \forall x \in \mathrm{dom}(Q), Q(x) \in \mathrm{Texp}[TQ,B]$$

Then we define Plexp to be a family of minimal sets of expressions for every type and for every compatible pair of type and ordinary assignments. We denote by Plexp[TQ,Q,a] (abbreviated P[TQ,Q,a] in this definition) the set of expressions with type a in Texp[TQ,B] with respect to the assignments TQ and Q:

Let $c \in C$, $x \in V$, $M \in P[TQ,Q,a]$, $N \in P[TQ,Q,b]$, $a,b \in \mathrm{Texp}[TQ,B]$:

	other conditions	expressions	$\in$ Plexp[TQ,Q,___]
constants:		c	$QC(c)$
variables:		x	$Q(x)$
products:		$\langle M,N\rangle$	$a\times b$
	$\forall L \in P[TQ,Q,a\times b]$	$L.1$	a
		$L.2$	b
functions:			
	$\forall L \in P[TQ,(Q \mid x{:}a),b]$	$\lambda x{:}a.L$	$a\rightarrow b$
	$\forall L \in P[TQ,Q,a'\rightarrow b]$, a cnv a'	$L(M)$	b
polymorphic functions:	$\forall L \in P[(TQ \mid t{:}k),Q',a]$ where $Q'=Q \rceil \{x \mid Q(x) \in \mathrm{Texp}[TQ-t,B]\}$	$\Lambda \iota{:}k.L$	$\Delta t{:}k.a$
	$\forall L \in [TQ,Q,\Delta t{:}k.a]$, $b \in \mathrm{Texp}[TQ,k]$	$L[b]$	$\{b/t\}a$
disjoint unions:		$\underline{\text{mkl}}\ (M,b)$	$a+b$
		$\underline{\text{mkr}}\ (a,N)$	$a+b$
	$\forall L \in P[TQ,Q,a+b]$, $f \in P[TQ,Q,a\rightarrow r]$, $g \in [TQ,Q,b\rightarrow r]$	$\underline{\text{case}}\ L\ \underline{\text{of}}\ (f,g)$	r
recursive types:	$\forall L \in P[TQ,Q,\mu t.a]$	$\underline{\text{unrec}}\ L$	$\{\mu t.a/t\}a$
	$\forall L \in P[TQ,Q,\{\mu t.a/t\}a$	$\underline{\text{mkrec}}\ L$	$\mu t.a$
recursive functions:	$\forall L \in P[TQ,Q,a\rightarrow a]$	$Y(L)$	a
conditional:			
	$\forall P \in P[TQ,Q,\mathrm{Bool}]$, $L \in P[TQ,Q,a]$	$\underline{\text{if}}\ P\ \underline{\text{then}}\ M\ \underline{\text{else}}\ L$	a

The most important language constructs to consider in defining the semantics will be products, functions, and polymorphic functions. The other constructs are included here for completeness and to show, briefly, that they can also be given semantics in a straightforward manner.

The two most important places in this definition where expressions are required to be "correctly-typed" are in the two forms of application. In ordinary application, the type of the argument must match the type of the bound variable, up to normal form in the type language. In the interests of generality, we decided not to restrict type applications to just base types, so that polymorphic functions can be applied to any types, including type generators. Here, then, for correctly-typed expressions, we require that the kind of the type argument match that of the bound type variable.

It is important to note how type abstraction and application provide the typechecking features necessary for user-defined types. In this language, a correctly typed expression $(\Lambda t{:}k.M)[b]$ has the property that M is correctly typed outside of the context in which the type b is known. Essentially, we can consider t to be the name of a user-defined type and b to be the concrete representation of t. Although this expression will be computationally equivalent to the expression M with b substituted for t, it is <u>not</u> assumed that b is equivalent to t in the definition and correct typing of M.

In a type abstraction, $\Lambda t{:}k.M$, we require a condition, called the "safe" condition, that the body of the abstraction is correctly typed with respect to a restricted assignment Q'. Q' has its domain restricted to only those variables, x, whose type expressions are correctly defined without the type variable t. The consequence of this condition is that to form an expression $\Lambda t{:}k.M$, it must be that t is not free in the type Q(x) of any free variable x in M. This ensures that we do not bind a type variable occurring in the type of a free variable.

As in Texp, the domains of the type assignment TQ and the ordinary assignment Q must include all the free variables of any expression in Plexp correctly typed with respect to those assignments.

Plexp has both type and ordinary variables, and we define substitution for both sorts of variables. These substitutions will be used to define reduction for the two sorts of abstraction and application.

First we need a notation for a type substitution applied to every type expression in the image of an ordinary assignment:

$\{a/t\}Q$ is the assignment: $\forall x \in \mathrm{dom}(Q)$, $(\{a/t\}Q)(x) = \{a/t\}Q(x)$.

We denote by $\{a/t\}M$ the substitution of a type expression a in Texp[TQ,k] for all free occurrences of the type variable t in the expression M in Plexp[(TQ|t:a),Q,b]. We can show that $\{a/t\}M$ is in Plexp[TQ,$\{a/t\}Q$,$\{a/t\}b$]. (Note the effect of this substitution on the type structure in the assignment Q.)

We denote by $\{N/x\}M$ the substitution of an expression N in Plexp[TQ,Q,a] for all free occurrences of a variable x in the expression M in Plexp[TQ,(Q|x:a'),b], where a cnv a'. We can show that $\{N/x\}M$ is in Plexp[TQ,Q,b'], where b cnv b'.

<u>Definition:</u> Computation (Reduction) rules

(α) $\lambda x{:}a.M \ \triangleright\ \lambda y{:}a.\{y/x\}M$ where $y \notin \mathrm{dom}(Q) \cup \{x\}$

(α) $\Lambda t{:}k.M \ \triangleright\ \Lambda s{:}k.\{s/t\}M$ where $s \notin \mathrm{dom}(TQ) \cup \{t\}$

(β) $(\lambda x{:}a.M)\,(N) \ \triangleright\ \{N/x\}M$

(β) $(\Lambda t{:}k.M)\,[b] \ \triangleright\ \{b/t\}M$

(τ) $\langle M,N\rangle.1 \ \triangleright\ M$

(τ) $\langle M,N\rangle.2 \ \triangleright\ N$

(There are other computation rules for the other language constructs that are not included here since they are not crucial in defining the part of the semantics that is newly treated with finitary retracts.)

If $M \ \triangleright\ N$ and $M \in$ Plexp[TQ,Q,a], then $N \in$ Plexp[TQ,Q,a'], where a cnv a'. It is known that every expression in this language does have a normal form, see

[Stenluna 1972].

III. The finitary domains and their continuous functions

The construction of mathematical structures appropriate for our model starts with a nonempty partially ordered set (poset). The important distinguishing characteristics will be which upper bounds (ub's) and least upper bounds (lub's) are present, whether the elements of the poset are "algebraic" with respect to some set of elements, and which bounds are preserved by the functions. These definitions and most of the material in this and the succeeding section are taken from [Scott 1980]. For this reason, the proofs are given in the appendix.

We will denote the partial ordering on a poset, D, by $\leq$. If the lub of a subset $X \subseteq D$ exists, it is denoted by $\sqcup X$.

Definition: An element $e \in D$ is *finite* iff for any subset $X \subseteq D$, if $e \leq \sqcup X$ then there exists a finite set $Xf \subseteq X$ such that $e \leq \sqcup Xf$.

Notation: The set of finite elements of D is denoted by E.

Definition: A subset $X \subseteq D$ is *consistent* iff every finite subset has an ub in D.

Definition: A subset $X \subseteq D$ is *directed* iff every finite subset has an ub in X, (or equivalently, if X is nonempty and every $x, y \in X$ has an ub in X).

Now we can define finitary domains to be nonempty poset's where all the consistent lub's exist and where all the elements are "algebraic" with respect to the finite elements:

A nonempty poset is a *finitary domain* iff

i) every consistent subset has a lub

and ii) every element is the lub of some finite elements.

As a consequence of the second condition of the definition, we have that

$$\forall x \in D,\ x = \sqcup \{e \in E \mid e \leq x\}.$$

This is the condition that we will often refer to by saying that a finitary domain D is algebraic. Note that every finitary domain has a least element, $\perp$, since the empty set is consistent and $\sqcup\emptyset=\bot$.

For the functions over the finitary domains, we take the usual notion of continuous functions, i.e. the functions preserve the lub's of all directed sets. We also establish other properties of the functions, including the crucial one that the set of continuous functions between two finitary domains is also a finitary domain.

Definition: A continuous function from D to D' is a function f such that

$$\text{for all directed sets } S\subseteq D,\ f(\sqcup S)=\sqcup f(S).$$

Proposition: A function f from D to D' is continuous iff

$$\forall e'\in E', x\in D\text{:}\ \ e'\le f(x) \text{ iff } \exists e\in E.\ e\le x \text{ and } e\le f(e)\ .$$

It follows from this proposition, that a continuous function f is completely determined by the set of finite pairs (e,e') such that $e'\le f(e)$. (Since D' is finitary, $f(x) = \sqcup\{e' \mid e'\le f(x)\} = \sqcup\{e' \mid \exists e\in E.\ e\le x \text{ and } e'\le f(e)\}$.)

Definition: The poset $D\rightarrow D'$ has as elements all continuous functions, f, from D to D' with the pointwise ordering: $f\le g$ iff $\forall x\in D.\ \ f(x)\le g(x)$.

Theorem: If D and D' are finitary domains, then $D\rightarrow D'$ is also.

In defining a model in a later section, we will be exploiting the close connection between models of a typed λ-calculus and cartesian closed categories. This connection will be explained in detail later, but for now, we find it useful to define this idea of cartesian closed category and show how the finitary domains form one.

First of all, we take the finitary domains to be the *objects* of a category and their continuous functions to be the *morphisms.* These objects and morphisms do form a category since identity functions are continuous and function composition preserves continuity.

The first rule of a cartesian closed category (c.c.c.) is that it have finite products. We define products here for the case with two objects.

<u>Definition:</u> Categorical product

For every pair of objects D and D', there is an object $D\times D'$ and morphisms $p:D\times D'\rightarrow D$ and $p':D\times D'\rightarrow D'$ such that for any object C and morphisms $f:C\rightarrow D$ and $f':C\rightarrow D'$, $\exists$ a unique morphism $<f,f'>$ such that the following diagram commutes:

$$\begin{tikzcd}
 & C \arrow[ld, "f"'] \arrow[d, "<f{,}f'>"] \arrow[rd, "f'"] & \\
D & D\times D' \arrow[l, "p"] \arrow[r, "p'"'] & D'
\end{tikzcd}$$

Furthermore, from the product on objects, one always obtains a product on morphisms: for all morphisms $f:C\rightarrow C'$ and $g:D\rightarrow D'$, there is a morphism $f\times g:C\times C'\rightarrow D\times D'$ such that $\forall z\in C\times C'$, $(f\times g)(z) = <f(p(z)),g(p'(z))>$.

Now in our case, we can define $D\times D'$ to be the usual pairs of elements:

<u>Definition:</u> $D\times D'=\{<x,x'>\mid x\in D,x'\in D'\}$, where $<x,x'>\leq<y,y'>$ iff $x\leq x'$ and $y\leq y'$.

<u>Proposition:</u> If D and D' are finitary domains, then so is $D\times D'$, where the set of finite elements is $E\times E'$.

<u>Theorem:</u> Let $<x,x'>\in D\times D'$. Define $p(<x,x'>)=x$ and $p'(<x,x'>)=x'$.

If C is any other finitary domain, and $f:C\rightarrow D$ and $f':C\rightarrow D'$ are continuous functions, define $\forall x\in C$, $<f,f'>(x)=<f(x),f'(x)>$. Then $D\times D'$ is the categorical product.

This 2-ary product can easily be extended to an n-ary product and to a possibly infinite product indexed by the elements of a countable set. The 0-ary product is just $\{\bot\}$, the terminal object in this category.

Now the second rule of a cartesian closed category is that function spaces are also objects in the category and that they interact with the product spaces in such a way that "currying" of function arguments works properly.

<u>Definition:</u> A category is cartesian closed iff it has finite products and for every pair of objects D and D', there is an object $D=>D'$ and a morphism

$ap:(D{=}{>}D')\times D\rightarrow D'$ such that for every object C and morphism $f:C\times D\rightarrow D'$, $\exists$ a unique morphism $h:C\rightarrow(D{=}{>}D')$ such that the following diagram commutes:

$$\begin{tikzcd}
C\times D \arrow[r, "h\times I"] \arrow[dr, "f"'] & (D{=}{>}D')\times D \arrow[d, "ap"] \\
 & D'
\end{tikzcd}$$

The morphism h is usually defined as ab(f) for a function $ab:(C\times D\rightarrow D')\rightarrow(C\rightarrow(D{=}{>}D'))$.

In our case, we can use the continuous functions both for morphisms in the category and to construct the function spaces $D{=}{>}D'$. We will continue to use the double arrow in the c.c.c. whenever we wish to distinguish the two roles of the continuous functions. Furthermore, we have already shown that the continuous functions, now $D{=}{>}D'$, form a finitary domain.

Theorem: Let C, D, and D' be finitary domains.

Let $f\in D{=}{>}D'$ and $x\in D$. Define $ap(\langle f,x\rangle)=f(x)$.

Let $f\in C\times D\rightarrow D', x\in C, y\in D$. Define $ab(f)(x)(y)=f(\langle x,y\rangle)$.

Then the finitary domains and their continuous functions form a c.c.c.

As a final property of the general mathematical construction of finitary domains and their continuous functions, we observe that a finitary domain, D, does have a least fixed point operator, $Y:(D\rightarrow D)\rightarrow D$. That is, if $f:D\rightarrow D$, then the least fixed point of f is given by

$$Y(f) = \bigcup_{n=0}^{\infty} f^{n}(\bot).$$

The proof of this depends on the continuity of f and the existence of lub's of directed sets in D.

IV. The finitary retractions

In constructing the model for our language, the interpretaton of a type will be a function which picks out a certain subcollection of elements, i.e. the elements of that type will be those in the image of the function representing the type. In defining just which functions will represent types, we need to use the following two related ideas:

Definition: For finitary domains D and D', D is a *retract* of D' iff there exist continuous functions $i:D\to D'$ and $j:D'\to D$ such that $j\circ i=I$ on D (the identity function on D).

Definition: A *retraction* on a finitary domain D is a continuous function $r:D\to D$ such that $r\circ r=r$.

Proposition: If the image of the retraction r on D, denoted r(D), is a finitary domain, then it is a retract of D.

If the image r(D) of a retraction is a finitary domain, we will call r a *finitary retraction*. The entire collection of finitary retractions on a finitary domain D will be denoted *FRet(D)* . The model of our programming language will consist of a finitary domain to represent programming language expressions and finitary retractions to represent types. The remainder of this section will describe various properties of finitary retractions and of FRet(D) itself, including the crucial property that FRet(D) is a finitary domain and itself the image of a finitary retraction. This property fails for ordinary retractions but holds for closures (retractions r such that $r\geq I$ on $D\to D$). The model given in Section VI depends on this property to give meanings to the types of polymorphic functions.

Although the image of an arbitrary retraction is not necessarily finitary, it does have lub's of all consistent subsets. The condition that fails is that not all elements of the image of a retraction may be algebraic (see [Scott 1980] for an example of this).

If r is a retraction on a finitary domain D, then

Lemma: $r(D) = \mathrm{Fix}(r) \equiv \{x\in D \mid r(x)=x\}$.

Lemma: r(D) has lub's of all consistent subsets. Furthermore, r(D) is *directed complete*, i.e. if X is a directed set, $X\subseteq r(D)$, then $\sqcup X$ in $r(D) = \sqcup X$ in D.

In establishing the principal characterization of finitary retractions, we must first investigate the functions that turn out to be the finite elements of FRet(D).

Definition: A retraction of a finitary domain D is a *finite retraction* iff it is a finite element of $D \to D$.

Note that a finite retraction also must be finitary, since the image of a finite function is always finite.

Definition: A continuous function f is a *preretraction* iff $f \leq f \circ f$.

Lemma:

a. If f is a finite preretraction, then the least retraction that it approximates, denoted ∇f, is also finite.

b. The finite retractions appromimating a given retraction form a directed set.

Theorem: Let $r: D \to D$ be a retraction. Then the following are equivalent:

i) r is finitary.

ii) $r(x) = \sqcup \{ r(e) \mid \exists e \in E . e \leq r(x) \text{ and } e \leq r(e) \}$.

(Note that this lub is the same in r(D) as in D since r(D) is directed complete.)

iii) r is the lub of some finite retractions.

Conditions i) and iii), of course, are crucial in showing that FRet(D) is algebraic with respect to the finite retractions. We also wish to show that FRet(D) is the image of a retraction on the finitary domain $D \to D$.

Definition: Let $\rho : (D \to D) \to (D \to D)$ be defined as the function such that $\forall g \in D \to D$:

$$\rho(g) = \sqcup \{ \nabla f \mid f \in E \to E . f \leq g \text{ and } f \leq f \circ f \}.$$

Lemma: If $r \in \mathrm{FRet}(D)$, then $\rho(r) = r$.

Theorem: FRet(D) is a finitary domain with the finite retractions as its finite elements and FRet(D) is a retract of $D \to D$ by the function ρ.

V. A calculus of finitary retracts

The next step in constructing a model for the language is to show how to calculate finitary retracts that will represent the type constructors. We will assume that the basis of the model is a finitary domain U that at least has the property that it solves a recursive domain equation with its own function and product spaces. Then we will show how to construct semantic types as finitary retracts over the domain U.

Definition: Let U be a finitary domain with U→U and U×U as retracts:

$$\begin{tikzcd}
 & U\to U \arrow[dl, bend right, "i"'] \\
U \arrow[ur, bend right, "j"'] \arrow[dr, bend right, "k"'] & \\
 & U\times U \arrow[ul, bend right, "l"']
\end{tikzcd}$$

where

$j\circ i=I$ on U→U

$k\circ l=I$ on U×U

That U→U and U×U are retracts of U means that U contains isomorphic copies of these domains. (Actually, as long as U has at least two finite elements, it would be sufficient to have U→U as a retract of U. Then U is known as a reflexive domain, and it can be shown that U×U is also a retract of U.)

Under these assumptions about the domain U, FRet(U) is also a retract of U, since it is contained in U→U via the retraction ρ.

$$\begin{tikzcd}
U \arrow[r, bend right, "j"'] & U\to U \arrow[l, bend right, "i"'] \arrow[r, bend right, "\rho"'] \arrow[r, phantom, "\geq"] & \mathrm{FRet}(U) \arrow[l, bend right, "I"']
\end{tikzcd}$$

Actually, we will use the isomorphic copy of FRet(U) that is contained in U for our domain of types. Denote this domain by $\underline{F}$Ret(U) and $\underline{\rho}$:U→U for the finitary retraction on U whose image is $\underline{F}$Ret(U). ($\underline{\rho}$ can be defined by: $\forall x\in U.\ \underline{\rho}(x) = i(\rho(j(x)))$.) Note that whenever we want to encode a function f:U→U as an element of U, we take i(f), and whenever we want to use an element x of U as a function, we take j(x). The meaning of the retract equation is, of course, that j(i(f))=f.

However, in order to simplify many equations, if $r \in \underline{F}\mathrm{Ret}(U)$, we will not always write $j(r)$ when the context shows that r is being using as a function, namely in function application or composition and in showing that $j(r)$ is a finitary retraction. Also, we will denote the image of $j(r)$ as $r(U)$, instead of $j(r(U))$. Note that since $\underline{\rho}$ is not only an element of $\underline{F}\mathrm{Ret}(U)$ but also $\underline{F}\mathrm{Ret}(U)=\underline{\rho}(U)$, we have $\underline{\rho}(\underline{\rho})=\underline{\rho}$. That $\underline{\rho}$ is in its own image (up to isomorphism) is what we mean intuitively by "the type of all types is a type".

First, we make explicit the remark that a retract of a domain D is isomorphic to a subdomain of D, and we show a useful result for defining continuous functions on the images of retractions by restricting the domain and corestricting the codomain of a continuous function on the whole domain.

Proposition: If D and D' are finitary domains and D is a retract of D' via the functions $i:D\to D'$ and $j:D'\to D$, then $D \simeq i(D)$.

Proof: We already have that $j \circ i = I$ on D. Let $x \in D'$ such that $x \in i(D)$. We must show that $i(j(x))=x$. But $x=i(y)$ for some $y \in D$. So $i(j(x))=i(j(i(y)))=i(y)=x$.□

Lemma: Let R and S be directed complete subdomains of U. Let $f:U\to U$ be such that for all $x \in R$, $f(x) \in S$. Then $f'=f \rceil R \lceil S \in R\to S$, i.e. the function f restricted to R and corestricted to S is a continuous function with domain R and codomain S.

Proof: The only problem is whether f' is continuous, so we must show that for all directed sets $X \subseteq R$, $f'(\cup X) = \cup f'(X)$, where the lub's are in R and S, respectively. But since R and S are directed complete, these are the same lub's as in U, and since f' has the same definition as f on the subdomain R, the equation holds. □

We now turn our attention to interpreting all type constructors as elements of $\underline{F}\mathrm{Ret}(U)$. We begin by showing that functions and products can be constructed as finitary retracts, and by showing that $\underline{F}\mathrm{Ret}(U)$ forms a cartesian closed category. The objects in the category will be elements r and s in $\underline{F}\mathrm{Ret}(U)$ and morphisms $r \circ\!\!\rightarrow s$ will be continuous functions from $r(U)$ to $s(U)$.

__Product Definition:__ Let $r,s \in \underline{F}\mathrm{Ret}(U)$. Let $r\otimes s$ be the encoding by i of the function that $\forall x \in U$ gives $l(<r(p(k(x))),\ s(p'(k(x)))>)$. This function is a continuous one in $U \rightarrow U$ from the definition. (Recall that $<\_,\_>$ is the tupling operator on arbitrary finitary domains, and that p and p' are the projection functions.)

__Proposition:__ $r\otimes s(U) \simeq r(U)\times s(U)$.

__Proof:__

Recall the retraction pair k,l from $U \underset{k}{\overset{l}{\leftrightarrows}} U\times U$. (So $k \circ l = I$ on $U\times U$.)

Let $k' : r\otimes s(U) \rightarrow r(U)\times s(U) = k\rceil r\otimes s(U) \lceil r(U)\times s(U)$.

For this to be a proper restriction and corestriction, we must have that $\forall x$ in $r\otimes s(U)$, $k(x)$ is in $r(U)\times s(U)$. A priori, $k(x)=<y,z>$, for some pair $<y,z>$ in $U\times U$. But $x \in r\otimes s(U)$, so

$$k(x) = k(r\otimes s(x)) = k(l(<r(p(k(x))),s(p'(k(x)))>))$$

$$= <r(p(k(x))),s(p'(k(x)))> .$$

So the first component is in $r(U)$ and the second is in $s(U)$.

Let $l' : r(U)\times s(U) \rightarrow r\otimes s(U) = l\rceil r(U)\times s(U) \lceil r\otimes s(U)$.

For this to be a proper restriction and corestriction, we must have that $\forall z$ in $r(U)\times s(U)$, $l(z)$ is in $r\otimes s(U)$.

$$r\otimes s(l(z)) = l(<r(p(k(l(z)))),s(p'(k(l(z))))>)$$

$$= l(<r(p(z)),s(p'(z))>)$$

But if z is in $r(U)\times s(U)$ then $p(z)$ is in $r(U)$, so $r(p(z))=p(z)$, and similarly, $s(p'(z))=p'(z)$. So the above equations are

$$= l(<p(z),p'(z)>)$$

$$= l(z).$$

We have shown that k' and l' are continuous functions with the right domains and codomains. Finally, we observe that $k'\circ l' = I$ since $k\circ l = I$. Then to show that $l'\circ k' = I$, let $x\in r\otimes s(U)$:

$l(k(x)) = l(k(r\otimes s(x))) = l(k(l(<r(p(k(x))),s(p'(k(x)))>))) = x.$ □

Proposition: If $r,s\in \underline{F}\mathrm{Ret}(U)$, then $r\otimes s\in \underline{F}\mathrm{Ret}(U)$.

Proof:

I. Show that $r\otimes s$ is a retraction by showing $(r\otimes s)\circ(r\otimes s)=r\otimes s$. Let $x\in U$:

$$\begin{aligned} r\otimes s(r\otimes s(x)) &= l(<r(p(k[l(<r(p(k(x)),s(p'(k(x)))>)]),s(p'(k[\ldots]))>) \\ &= l(<r(r(p(k(x)))),s(s(p'(k(x)))>) \\ &= r\otimes s(x) \end{aligned}$$

II. The image of $r\otimes s$ is finitary since it is isomorphic to the finitary domain $r(U)\times s(U)$.

Proposition: $r\otimes s$ is a categorical product.

Proof: We have already shown that for any $r,s\in \underline{F}\mathrm{Ret}(U)$, we can define an object $r\otimes s\in \underline{F}\mathrm{Ret}(U)$. Now we have to define projection morphisms, $\underline{p}:r\otimes s\to r$ and $\underline{p}':r\otimes s\to s$ such that if $q\in \underline{F}\mathrm{Ret}(U)$, $f:q\to r$, and $f':q\to s$, there is a unique morphism $h:q\to r\otimes s$ such that the following diagram commutes:

$$\begin{tikzcd}
 & q \arrow[ld, "f"'] \arrow[d, "h"] \arrow[rd, "f'"] & \\
r & r\otimes s \arrow[l, "\underline{p}"] \arrow[r, "\underline{p}'"'] & s
\end{tikzcd}$$

(We have used the "name" of a retraction in place of its image, e.g. r instead of $r(U)$, to simplify the diagram). But, given the isomorphism between $r\otimes s(U)$ and $r(U)\times s(U)$, for some isomorphism functions k' and l', (there is actually a different k' and l' for every r and s), this can all be defined in terms of products on finitary domains:

$$\begin{tikzcd}
& q \arrow[dl, "f"'] \arrow[d, "{<f,f'>}"] \arrow[dr, "f'"] & \\
r & r\times s \arrow[l, "p"'] \arrow[r, "p'"] \arrow[d, bend right, "l'"'] & s \\
& r\otimes s \arrow[ul, "\underset{\sim}{p}"] \arrow[ur, "\underset{\sim}{p}'"'] \arrow[u, bend right, "k'"'] &
\end{tikzcd}$$

so that if $\underset{\sim}{p}$ and $\underset{\sim}{p}'$ are defined by $\underset{\sim}{p}=p\circ k'$, $\underset{\sim}{p}'=p'\circ k'$, then h is defined as $h=l'\circ<f,f'>$, which is unique up to isomorphism. □

We will also denote by $\otimes$ the product of continuous functions, which is (as noted before) defined pointwise using the projection functions, $\underset{\sim}{p}$ and $\underset{\sim}{p}'$ in this case.

Typed Functions

Definition: Let $r,s\in \underset{\sim}{F}Ret(U)$. Let $r\circ\!\!\to s$ be the encoding by i of the function that $\forall z$ in U gives $i(s\circ j(z)\circ r)$.

Proposition: $r\circ\!\!\to s(U) \simeq r(U)\to s(U)$

Proof:

Recall the retraction pair i,j from $U \underset{j}{\overset{i}{\rightleftarrows}} U\to U$. (So $j\circ i=I$ on $U\to.U$)

Let $j'\in(r\circ\!\!\to s)(U)\to(r(U)\to s(U))$ be the function such that $\forall z \in r\circ\!\!\to s(U)$, $j'(z) = j(z)\rceil r(U)\lceil s(U)$.

To show that this is a proper restriction and corestriction, we must show that $\forall x$ in $r(U)$, $j(z)(x)$ is in $s(U)$:

$$\begin{aligned}
s(j(z)(x)) &= s(j(i(s\circ j(z)\circ r))(x)) \\
&= s(s(j(z)(r(x)))) \\
&= s(j(z)(r(x))) \\
&= j(z)(x)\ .
\end{aligned}$$

The definition of the other isomorphism function is more complicated than a simple restriction and corestriction of i, since $r(U)\to s(U)$ is not a subset of $U\to U$. These two sets of functions don't have a subset relation because any function in $r(U)\to s(U)$ has a different domain and codomain than the functions in $U\to U$.

Let $i' \in (r(U)\to s(U)) \to (r\circ\!\!\to s)(U) = h\lceil r\circ\!\!\to s(U)$, where $h:(r(U)\to s(U))\to U$ is the function such that

$\forall g\in r(U)\to s(U),\ h(g) = i(I\rceil s(U)\circ g\circ r\lceil r(U))$.

(Note that $(I\rceil s(U)\circ g\circ r\lceil r(U))$ is a function in $U\to U$.)

The restriction of I to s(U) and the corestriction of r to r(U) are obviously both proper. We must show that the corestriction of h is proper by showing that $\forall$ g in $r(U)\to s(U)$, h(g) is in $r\circ\!\!\to s(U)$:

$$\begin{aligned} r\circ\!\!\to s(h(g)) &= i(s\circ j(h(g))\circ r) \\ &= i(s\circ j(i(I\rceil s(U)\circ g\circ r\lceil r(U)))\circ r) \\ &= i(s\circ I\rceil s(U)\circ g\circ r\lceil r(U)\circ r) \\ &= i(I\rceil s(U)\circ g\circ r\lceil r(U)) \\ &= h(g)\ . \end{aligned}$$

To show that $j'\circ i' = I$ on $r(U)\to s(U)$, let g be in $r(U)\to s(U)$:

$$\begin{aligned} j'(i'(g)) &= j(i(I\rceil s(U)\circ g\circ r\lceil r(U)))\rceil r(U)\lceil s(U) \\ &= (I\rceil s(U)\circ g\circ r\lceil r(U))\rceil r(U)\lceil s(U) \\ &= g\ ,\quad \text{since } \forall x\in r(U),\ g(x) = g(r(x)). \end{aligned}$$

To show that $i'\circ j' = I$ on $r\circ\!\!\to s(U)$, let z be in $r\circ\!\!\to s(U)$:

$$\begin{aligned} i'(j'(z)) &= i(I\rceil s(U)\circ(j(z)\rceil r(U)\lceil s(U))\circ r\lceil r(U)) \\ &= i(s\circ j(z)\circ r) \\ &= r\circ\!\!\to s(z) \\ &= z.\quad \square \end{aligned}$$

<u>Proposition:</u> If $r,s \in \underline{F}\mathrm{Ret}(U)$, then $r\circ\!\!\to s \in \underline{F}\mathrm{Ret}(U)$.

<u>Proof:</u>

I. Show that $r\circ\!\!\to s$ is a retraction by showing $(r\circ\!\!\to s)\circ(r\circ\!\!\to s) = (r\circ\!\!\to s)$.

Let $z\in U$: $r^\circ\!\to s(r^\circ\!\to s(z))=i(s\circ j[i(s\circ j(z)\circ r)]\circ r)$

$$=i(s\circ s\circ j(z)\circ r\circ r)$$

$$=r^\circ\!\to s(z)\ .$$

II. The image of $r^\circ\!\to s$ is finitary since it is isomorphic to the finitary domain $r(U)\to s(U)$.

Theorem: $\underline{F}\mathrm{Ret}(U)$ with continuous functions is a cartesian closed category.

Proof: 1) It has finite products: n-ary products are easily extended from $r\otimes s$, and the 0-ary product is the encoding of the constant function that always gives $\perp$, which is a finitary retraction.

2) We have already shown that $\forall r,s\in\underline{F}\mathrm{Ret}(U)$, we can define an object $r^\circ\!\to s\in\underline{F}\mathrm{Ret}(U)$. Now we have to define a continuous function $\underline{a}p:((r^\circ\!\to s)\otimes r)\to s$ such that $\forall q\in\underline{F}\mathrm{Ret}(U)$ and functions $f:q\otimes r\to s$, there exists a unique function $h:q\to(r^\circ\!\to s)$ such that the following diagram commutes:

$$\begin{tikzcd}
q\otimes r \arrow[r, "h\otimes I"] \arrow[dr, "f"'] & (r^\circ\!\to s)\otimes r \arrow[d, "\underline{a}p"] \\
 & s
\end{tikzcd}$$

where there is a continuous function $\underline{a}b:((q\otimes r)^\circ\!\to s)\to(r^\circ\!\to s)$ such that $h=\underline{a}b(f)$. But, given the isomorphism functions i and j between $r^\circ\!\to s(U)$ and $r(U)\to s(U)$, this can all be defined in terms of the functions ab and ap on ordinary finitary domains:

$$\begin{tikzcd}
q\otimes r \arrow[r, bend right, "k'"'] \arrow[drr, "f"'] & q\times r \arrow[l, bend right, "l'"'] \arrow[r, "ab(f\circ l')\times I"] \arrow[dr, "f\circ l'"] & (r\to s)\times r \arrow[d, "ap"] \arrow[r, bend right, "l''\circ(i'\times I)"'] & (r^\circ\!\to s)\otimes r \arrow[l, bend right, "(j'\times I)\circ k''"'] \arrow[dl, "\underline{a}p"] \\
 & & s &
\end{tikzcd}$$

So the function $\underline{a}p$ can be defined by $\underline{a}p=ap\circ(j'\times I)\circ k'$.

We are using k' and l' for the isomorphism functions between $q\otimes r$ and $q\times r$, and k'' and l'' for those between $(r^\circ\!\to s)\otimes r$ and $(r^\circ\!\to s)\times r$. Now, for any function $g\in$

$q \to (r \circ\!\!\to s)$ and $g' \in r \to r$, $g \otimes g' = l'' \circ (g \times g') \circ k'$. So $\underline{a}b(f)$ must be defined as $i' \circ ab(f \circ l')$, since then for $h = \underline{a}b(f)$, we have $h \otimes I = l'' \circ ((i' \circ ab(f \circ l')) \times I) \circ k'$, which is the composition of the functions that make this diagram commute. □

"Every c.c.c. is a model of the typed λ-calculus"

Although this theorem has been widely quoted, there is not uniform agreement about what the typed λ-calculus is and what a model for it should be. (The main differences over the former are about the relationship of types to ordinary variables and the changing of bound variables during substitution. The main differences over the latter are essentially over what forms of extensionality should be in the model; see [Berry 1980] for a discussion of this.) Therefore, we want to make explicit how a c.c.c. is a model for our typed λ-calculus subset of Plexp. This version of the theorem is taken from [Oles 1982] and is presented here as a special case of a more complicated semantics involving states as well as environments.

For convenience, we use the following subsets of Texp and Plexp as syntax for the typed λ-calculus with products:

type constants: $tc \in \mathrm{Texp}[TQ,B]$

products: $a \times b \in \mathrm{Texp}[TQ,B]$

functions: $a \to b \in \mathrm{Texp}[TQ,B]$

There are no type variables in this subset so we can assume that the type assignment TQ is arbitrary.

variables: $x \in \mathrm{Plexp}[TQ,Q,Q(x)]$

products: $\langle M,N\rangle \in \mathrm{Plexp}[TQ,Q,a \times b]$

$L.1 \in \mathrm{Plexp}[TQ,Q,a]$

$L.2 \in \mathrm{Plexp}[TQ,Q,b]$

functions: $\lambda x{:}a.L \in \mathrm{Plexp}[TQ,Q,a \to b]$

$L(M) \in \mathrm{Plexp}[TQ,Q,b]$

These definitions are all made under the same assumptions for a, b, L, M, and N as in the previous definition in section II.

Assume that we have a c.c.c., K, using the notations ×, p, p', <_,_>, ap, and ab as defined in section III. First, we give a meaning function for type expressions:

Mt: Texp[TQ,B] → objects in K.

Mt[[tc]] = kc, some object in K

Mt[[a×b]] = Mt[[a]] × Mt[[b]]

Mt[[a→b]] = Mt[[a]] => Mt[[b]]

Then, we give a family of meaning functions for Plexp, one for every set of typed expressions for some type a:

Me: Plexp[TQ,Q,a] → (E → Mt[[a]])

where E is the set of environment functions whose domain is the domain of Q such that ∀x∈dom(Q), e(x) ∈ Mt[[Q(x)]]. It is required that E also be an object in the c.c.c.

Me[[x]]e = e(x)

* Me[[<M,N>]]e = <Me[[M]]e, Me[[N]]e>

Me[[L.1]]e = p(Me[[L]]e)

Me[[L.2]]e = p'(Me[[L]]e)

Me[[λx:a.L]] = ab(f), where f is that function (morphism)
from E×Mt[[a]] to Mt[[b]] , such that
f(e,u) = Me[[L]](e|x→u).

Me[[L(M)]] = ap(Me[[L]]e, Me[[M]]e)

Equation * has been simplified using the pointwise nature of our definition of function tupling. "(e|x→u)" denotes the environment such that ∀y∈dom(Q|x:a), (e|x→u)(y) = if y=x then u else e(y).

The proof that these semantic equations give a model for the typed λ-calculus uses the following instance of the c.c.c. abstraction and application diagram:

$$\begin{tikzcd}
E \times Mt[[a]] \arrow[rr, "ab(f) \times I"] \arrow[dr, "f"'] & & Mt[[a\circ\!\!\to b]] \times Mt[[a]] \arrow[dl, "ap"] \\
& Mt[[b]] &
\end{tikzcd}$$

This shows why the set of environments E must be an object in the c.c.c.

That the c.c.c. is a model for this typed λ-calculus means that the semantics is correctly typed:

if $L \in \mathrm{Plexp}[TQ,Q,a]$ *then* for all environments whose domain is Q and whose codomain is correctly typed as defined above, $Me[[L]]e \in Mt[[a]]$.

We can also verify that the computation rules are preserved for this subset, which means that for α and β-reduction for ordinary functions and the two product reductions, we have:

if $M,N \in \mathrm{Plexp}[TQ,Q,a]$ such that $M \triangleright N$, *then* $Me[[M]] = Me[[N]]$.

(It is also the case that the c.c.c. semantics preserves the computation rule for η-reduction:

(η) $M \triangleright \lambda x{:}a.M(x)$, where x does not occur free in M.)

Another reason for introducing this theorem about c.c.c.'s is that we can now, without further argument, use λ-expressions to denote typed continuous functions over $\underline{F}\mathrm{ret}(U)$.

Polymorphic Functions

Definition: Let $a \in \underline{F}\mathrm{Ret}(U)$, $\alpha \in U \rightarrow U$ such that if $x \in a(U)$, then $\alpha(x) \in \underline{F}\mathrm{Ret}(U)$. Then let $\Delta(\alpha,a)$ be the encoding of the function: $\lambda z{:}U.i(\lambda t{:}U.\alpha(a(t))(j(z)(a(t))))$.

The intuitive interpretation of this retraction is that z is being using as a polymorphic function, the retraction a makes the argument to z into a (correctly-typed) type, and the function α makes the result of the polymorphic function have some type which is also dependent on the type argument. We can formalize this as a kind of infinite cartesian product.

Definition: Let T be a countable set and let F be a function which maps an element of T into a set, then

$$\prod_{t \in T} F(t)=\{\text{functions } f \mid \text{domain of } f \text{ is } T \text{ and } \forall t \in T,\ f(t) \in F(t)\}.$$

(This is a cartesian product indexed by the possibly infinite set T, where each function can be interpreted as an infinite product of all the second components of the graph of the function, and for each $t \in T$, $f(t)$ is the projection mapping for the component at t.)

The crucial property of this model - that $\underline{F}\mathrm{Ret}(U)$ itself be the image of a retraction - is used in the definition of polymorphic functions in the assumption that there is a retraction a which makes the argument of a polymorphic function into a type. For example, when base types are modeled by $\underline{F}\mathrm{Ret}(U)$, a polymorphic function that can accept any base type will use $\underline{\rho}$ for a, since $\underline{F}\mathrm{Ret}(U)$ is the image of $\underline{\rho}$.

<u>Proposition:</u> $\Delta(\alpha,a)(U) \simeq \prod_{t\in a(U)} \alpha(t)(U)$.

<u>Proof:</u>

Within this proof, we will denote the product over all $t\in a(U)$ by just $\Pi\alpha(t)(U)$.

Recall the retraction pair $U \overset{i}{\underset{j}{\rightleftarrows}} U\to U$, where $j\circ i = I$ on $U\to U$.

As with type functions, we can't define the isomorphism function from $\Pi\alpha(t)(U)$ to $\Delta(\alpha,a)(U)$ by a simple restriction and corestriction of i since the functions in $\Pi\alpha(t)(U)$ have domain $a(U)$ and codomain U and are thus not a subset of $U\to U$.

Let $\Phi : \Pi\alpha(t)U) \to \Delta(\alpha,a)(U) = (\lambda g{:}\, \Pi\alpha(t)(U).i(g\circ a\lceil a(U)))\lceil\Delta(\alpha,a)(U)$.

Let $\Psi{:}\Delta(\alpha,a)(U) \to \Pi\alpha(t)(U) = \lambda z{:}\Delta(\alpha,a)(U).j(z)\rceil a(U)$.

1. First, we must show that the definition of Φ fits the restriction and corestriction lemma. The corestriction of a to $a(U)$ is obviously proper; to show that the other corestriction is proper, we must show that $\forall g\in \Pi\alpha(t)(U)$ that $(\lambda g{:}\,\Pi\alpha(t(U).i(g\circ a\lceil a(U))(g) \ \in \ \Delta(\alpha,a)(U)$.

$$\begin{aligned}\Delta(\alpha,a)(i(g\circ a\lceil a(U))) &= i(\lambda t{:}U.\alpha(a(t))(j(i(g\circ a\lceil a(U)))(a(t)))\\ &= i(\lambda t{:}U.\alpha(a(t))(g\circ a\lceil a(U))(a(t)))\\ &= i(\lambda t{:}U.(g\circ a\lceil a(U)\circ a)(t)), \quad \text{since } g\in\Pi\alpha(t)(U)\\ &= i(g\circ a\lceil a(U)), \text{ by } \eta\text{-conversion.}\end{aligned}$$

2. The corestriction in the definition of Ψ is obviously proper.

3. Show that $\Psi\circ\Phi = I$ on $\prod\alpha(t)(U)$. Let $g \in \prod\alpha(t)(U)$:

$$\Psi(\Phi(g)) = (j(i(g\circ a\lceil a(U))))\rceil a(U)$$

$$= (g\circ a\lceil a(U))\rceil a(U)$$

$$= g, \text{ since } \forall t\in a(U),\ g(a(t))=g(t) \text{ and } dom(g)=a(U).$$

4. Show that $\Phi \circ \Psi = I$ on $\Delta(\alpha,a)(U)$. Let $z\in\Delta(\alpha,a)(U)$:

$$\Phi(\Psi(z)) = i((j(z)\rceil a(U))\circ a\lceil a(U))$$

$$= i(j(z)\circ a)$$

$$= i(j(\Delta(\alpha,a)(z))\circ a)$$

$$= i(j(i(\lambda t{:}U.\alpha(a(t))(z(a(t)))))\circ a)$$

$$= i(\lambda t{:}U.\alpha(a(t))(z(a(a(t)))))$$

$$= i(\lambda t{:}U.\alpha(a(t))(z(a(t))))$$

$$= \Delta(\alpha,a)(z)$$

$$= z.$$

<u>Proposition:</u> Let $a\in\underline{F}Ret(U)$ and $\alpha\in U\to U$ such that if $x\in a(U)$ then $\alpha(x)\in\underline{F}Ret(U)$. Then $\Delta(\alpha,a)\in\underline{F}Ret(U)$.

<u>Proof:</u>

I. Show that $\Delta(\alpha,a)$ is a retraction by showing $\Delta(\alpha,a)\circ\Delta(\alpha,a) = \Delta(\alpha,a)$.

Let $z\in U$. $\Delta(\alpha,a)(\Delta(\alpha,a)(z))$

$$=i(\lambda t{:}U.\alpha(a(t))(j[i(\lambda t{:}U.\alpha(a(t))(j(z)(a(t))))](a(t))))$$

$$=i(\lambda t{:}U.\alpha(a(t))(\alpha(a(a(t))(j(z)(a(a(t))))$$

$$=\Delta(\alpha,a)(z)$$

II. The proof that products indexed by 2 objects is finitary can easily be extended to show that products indexed by an arbitrary set is finitary. Then the image of $\Delta(\alpha,a)$ is finitary because it is isomorphic to the infinite product indexed by the set $a(U)$: $\prod_{t\in a(U)}\alpha(t)(U)$.

Recursive definitions:

Due to the special properties of $\underline{F}\mathrm{Ret}(U)$, we can interpret both recursive function definition and recursive type definition as a least fixed point operator on the image of a retraction (both function spaces and types are represented as the images of retractions). Actually, since these domains are directed complete, we can prove that these fixed points are given by the usual least fixed point operator on the whole domain, U:

$$Y:(U\to U)\to U = \lambda f:U\to U.\ \bigcup_{n=0}^{\infty} f^{n}(\bot).$$

Proposition: Let $r\in\underline{F}\mathrm{Ret}(U)$. Let $f\in U\to U$ such that $i(f)\in r\circ\!\!\to r(U)$. Let $f'=f\rceil r(U)\lceil r(U)$. Then $Y(f)$ is the least fixed point of f' over $r(U)$.

Proof: Note that f' is continuous by the restriction and corestriction lemma. We prove that f and f' have the same least fixed point by showing that every fixed point of f is a fixed point of f', and vice versa.

a. If $z\in r(U)$ is a fixed point of f', then $f=f'$ on $r(U)$ so

$f(z)=f'(z)=z$.

b. If $z\in U$ is a fixed point of f, first we must show that $z\in r(U)$:

$$r(z)=r(f(z))=r(r\circ\!\!\to r(f(z)))=r\circ\!\!\to r(f(z))=z.$$

Then, again $f=f'$ on $r(U)$, so $f'(z)=f(z)=z$. □

We omit here the definition of semantic constructs for disjoint unions and conditional, since the details of these work out similarly as on other c.p.o.'s or complete lattices and are not particularly dependent on the finitary retracts.

VI. The Model

A model for our language will consist of giving meanings for every expression in the three languages - Plexp, Texp, and Kexp, and showing that the semantic meanings obey the same typing and computation rules that the syntactic expressions do. The basic structure of the model is that expressions in Plexp will be interpreted as elements of the domain U, expressions in Texp as finitary retracts on U, i.e. elements of $\underline{F}$Ret(U), and expressions in Kexp as finitary retracts on $\underline{F}$Ret(U).

Since Plexp is an extension of the typed λ-calculus, the proof that this structure is a model will be based upon the theorem that "every cartesian closed category is a model of the typed λ-calculus" and the fact that the finitary retracts on the finitary domain U form a cartesian closed category (c.c.c.). Furthermore, the main extension to the type language, Texp, was to include type generators by making it a typed λ-calculus also. Thus, we can use the c.c.c. theorem again to use the finitary retracts on the finitary domain $\underline{F}$Ret(U) as a model for Texp. Finally, we must show that we can model the main extension to the programming language - polymorphic functions and application.

Semantics for the type language

The type language is itself a typed λ-calculus, and we will use the c.c.c. of finitary retracts on the finitary domain $\underline{F}$Ret(U). The type language does not actually have products so that we just need the c.c.c. for semantics for functions from types to types. We add to the semantic function some finitary retracts for other type constructors which have no computation rules and thus require no more proof. Recall that we used $\circ\!\!\rightarrow$ to denote function spaces of finitary retracts and $\underline{ap}$ and $\underline{ab}$ for the c.c.c. functions on those spaces.

The meaning function for types of types is a function Mk:Kexp → $\underline{F}$Ret(U), since finitary retracts on $\underline{F}$Ret(U) are also embedded in $\underline{F}$Ret(U). Kexp has only one constant and the type of typed functions:

$$\mathrm{Mk}[[B]] = \underline{\rho}$$

$$\mathrm{Mk}[[k{=}{>}m]] = \mathrm{Mk}[[k]] \circ\!\!\rightarrow \mathrm{Mk}[[m]].$$

The meaning function for type expressions requires an environment function from a finite subset of type variables to finitary retracts. If we make a finite subset of type variables into a "flat" poset by attaching a $\perp$ element, then it is a finitary domain. Let Te[TQ] be the set of type correct type environment functions. These functions all have finite images and are thus finite functions on Te[TQ], so that it is also a finitary domain and can be defined as the image of a finitary retract on $\underline{F}$Ret(U).

$$\mathrm{Te}[TQ] = \{te \in \mathrm{dom}(TQ) \rightarrow \underline{F}\mathrm{Ret}(U) \mid \forall t \in \mathrm{dom}(TQ),\ te(t) \in \mathrm{Mk}[[TQ(t)]]\}.$$

So the meaning function for types is a function

$$\mathrm{Mt}[TQ,k]\text{: } \mathrm{Texp}[TQ,k] \rightarrow (\mathrm{Te}[TQ] \rightarrow \mathrm{Mk}[[k]])\text{:}$$

(We write Mt for Mt[TQ,k], since TQ and k are obvious from the context.)

$$\mathrm{Mt}[[t]]te = te(t)$$

$$\mathrm{Mt}[[ta]]te = \text{some finitary retract assigned to the constant } ta$$

$$\mathrm{Mt}[[a\rightarrow b]]te = \mathrm{Mt}[[a]]te \circ\!\!\rightarrow \mathrm{Mt}[[b]]te$$

$$\mathrm{Mt}[[a\times b]]te = \mathrm{Mt}[[a]]te \otimes \mathrm{Mt}[[b]]te$$

$$\mathrm{Mt}[[\mu t.a]]te = Y(\lambda d{:}\underline{\rho}.\ \mathrm{Mt}[[a]](te \mid t\rightarrow d)))$$

$$\mathrm{Mt}[[\Delta t{:}k.a]]te = \Delta(\mathrm{Mt}[[\lambda t{:}k.a]]te,\ \mathrm{Mk}[[k]])$$

$$\mathrm{Mt}[[\lambda t{:}k.a]] = \underline{a}b(\lambda\langle te,d\rangle\text{: } \mathrm{Te}[TQ \mid t{:}k] \otimes \mathrm{Mk}[[k]].\ \mathrm{Mt}[[a]](te \mid t\rightarrow d))$$

$$\mathrm{Mt}[[a[b]]]te = \underline{a}p(\mathrm{Mt}[[a]]te,\ \mathrm{Mt}[[b]]te)$$

Note that Mt is, in fact, type correct by definition. The equations for the four constructors →, ×, , and Δ do give meanings in Mk[[B]], which is just $\underline{F}$Ret(U). We should also remark that Mt[[a]] exists for all types a, since it is a family of functions with a mutually recursive definition over domains Te[TQ] → Mk[[k]], which are finitary domains.

The α and β -reduction rules are preserved due to the theorem that every c.c.c. is a model of the typed λ-calculus.

Semantics for the programming language

The programming language is an extension of the typed λ-calculus, and we will use the c.c.c. of finitary retracts on the finitary domain U to model the typed λ-calculus subset, including products in this case. We add to the semantic function meanings for polymorphic functions and application and must then prove that their α and β-reduction rules are preserved.

In the last section, we gave the appropriate finitary retracts on U as meanings of × and → in the type language to be the types of products and ordinary functions in this language.

The meaning function requires not only an environment for type variables but one for ordinary variables as well. Let $E[TQ,Q] = \{e \in dom(Q) \rightarrow \underline{F}Ret(U) \mid \forall x \in dom(Q),\ \forall te \in TE[TQ],\ e(x) \in Mt[[Q(x)]]te\}$ be the set of type correct environment functions. Then the meaning function is

$Me[TQ,Q,a]\colon\ Plexp[TQ,Q,a] \rightarrow (Te[TQ] \rightarrow (E[TQ,Q] \rightarrow Mt[[a]]te)))$:

(Again, we write Me for Me[TQ,Q,a] when TQ, Q, and a are obvious.)

$Me[[x]]te\ e = e(x)$

$Me[[c]]te\ e =$ some constant in $Mt[[QC(c)]]te$

$Me[[<M,N>]]te\ e = <Me[[M]]te\ e,\ Me[[N]]te\ e>$

$Me[[L.1]]te\ e = \underline{p}(Me[[L]]te\ e)$

$Me[[L.2]]te\ e = \underline{p}'(Me[[L]]te\ e)$

$Me[[\lambda x{:}a.L]]te = \underline{a}b(\lambda <e,\ u>{:}E[TQ,Q \mid x{:}a] \otimes Mt[[a]]te.\ Me[[L]]te\ (e \mid x \rightarrow u))$

$Me[[L(M)]]te\ e = \underline{a}p(Me[[L]te\ e,\ Me[[M]]te\ e)$

$Me[TQ,Q,\Delta t{:}k.a]\ [[\Lambda t{:}k.L]]te\ e = (\lambda d{:}Mk[[k]].$

$\qquad Me[TQ \mid t{:}k,\ Q',\ a]\ [[L]]\ (te \mid t \rightarrow d)e')$

where $e' = e \rceil dom(Q')$, and Q' was defined in Section I.

$Me\ [[\ L[a]\ ]]te\ e =\ (Me[[L]]te\ e)(Mt[[a]]te)$

Again, we note that the semantic function is type correct by definition and exists because it is a family of mutually recursive functions defined over finitary retracts. The typed λ-calculus subset fits the semantic definition for an arbitrary c.c.c. because for a fixed type environment te, $\mathrm{Mc}[[M]]te$ is just a function from environments to meanings.

To complete the proof that this semantic function gives a model for the language, we must show that the computation rules for polymorphic functions are preserved. First, we present a lemma that extension of environment functions correctly interprets substitution. This lemma is actually the crux of the modelling of computation rules, but we omit the proof here since it is a rather straightforward structural induction on terms. The full proof of a similar theorem appears in [McCracken 1979].

Lemma: Let $b \in \mathrm{Texp}[TQ,k]$ and $M \in \mathrm{Plexp}[TQ|t{:}k,Q,a]$ so that $\{b|t\}M$ is a well-defined substitution. If $te \in \mathrm{Te}[TQ]$ and $e \in E[TQ,Q]$, then

$$\mathrm{Me}[[\{b/t\}M]](te)(e) = \mathrm{Me}[[M]](te|t{\to}\mathrm{M}[[b]]te)(e).$$

(Note that two other lemmas would have been necessary for the other computation rules: namely, one of the form $\mathrm{Mt}[[\{b/t\}a]](te) = \mathrm{Mt}[[a]](te|t{\to}\mathrm{Mt}[[b]]te)$ for reduction of type generators, and one of the form $\mathrm{Me}[[\{N/x\}M]](te)(e)= \mathrm{Me}[[M]](te)(e)x{\to}\mathrm{Me}[[N]](te)(e))$ for ordinary reduction in Plexp.)

With this important lemma, we can complete the proof of the model.

Theorem : Let $M \in \mathrm{Plexp}[TQ,Q,a]$ and $N \in \mathrm{Plexp}[TQ,Q,a']$, where a cnv a', such that $M \triangleright N$. Then $\mathrm{Me}[[M]] = \mathrm{Me}[[N]]$.

Proof : Given our theorem about cartesian closed categories, we need only show that α and β-reduction for polymorphic functions are preserved. Let $te \in \mathrm{Te}[TQ]$, $e \in E[TQ,Q]$:

(α) Let $\Lambda t{:}k.M \in \mathrm{Plexp}[TQ,Q,\Delta t{:}k.a]$. Then $\Lambda s{:}k.\{s/t\}M \in \mathrm{Plexp}[TQ,Q,\Delta t{:}k.a']$, where a cnv a', and $s \notin \mathrm{dom}(TQ) \cup \{t\}$. By the definition,

$\mathrm{Me}[[\Lambda s{:}k.\{s/t\}M]](te)(e) = \Phi(\lambda d{:}\mathrm{Mk}[[k]].\mathrm{Me}[[\{s/t\}M]](te|s{\to}d)(e'))$.

Then by our substitution lemma, this is

$$\Phi(\lambda d{:}Mk[[k]].Me[[M]]((te|s\to d)|t\to Mt[[s]](te|s\to d))(e'))$$

$$= \Phi(\lambda d{:}Mk[[k]].Me[[M]]((te|s\to d)|t\to d).$$

Now we must be explicit that in this equation we are using Me[(TQ|s:k|t:k),Q,a], but since s does not occur free in M, we can also use Me[(TQ|t:k),Q,a], and then it is obvious from the definition of Me that the above equation

$$=\Phi(\lambda d{:}Mk[[k]].Me[[M]](te|t\to d)) = Me[[\Lambda t{:}k.M]](te). \ \square$$

(ρ) Let $(\Lambda t{:}k.M)[a] \in Plexp[TQ,Q,b]$ and $a \in Texp[TQ,k]$.

$$Me[[(\Lambda t{:}k.M)[a]]](te)(e) = \Psi(Me[[\Lambda t{:}k.M]](te)(e))(Mt[[a]](te)))$$

$$= \Psi(\Phi(\lambda d{:}Mk[[k]].Me[[M]](te|t\to d)(e')))(Mt[[a]](te))$$

(Note that this is a correctly formed application of a function on finitary retracts.)

$$= Me[[M]](te|t\to Mt[[a]](te))(e)$$

$$= Me[[\{a/t\}M]](te)(e), \text{ by the substitution lemma. } \square$$

Discussion

This type of model for polymorphic functions has the advantage that it_s setting is in domains which can model a useful variety of language features, including mutually recursive definitions (which were not included in this paper but can easily be added.) The main drawback is that the partial order among types modelled by finitary retracts (and also closures) doesn't seem to have any useful function. In particular, it doesn't lend itself to the definition of "representation relations", see [Reynolds 1974], that would help prove Reynolds' representation theorem. The present version of the model has more "polymorphic" functions than would satisfy such a theorem.

Other language features which have yet to be worked out in this kind of a language are coercions among types, "inheritance" of defined types, overloading of operators, and type deducing.

References

[Berry 1980]
Berry, Gerard, On the definition of lambda-calculus models. INRIA report no. 46, December 1980.

[Donahue 1979]
Donahue, James, On the semantics of "data type". Siam J. Comput., Vol. 8, No. 4, November 1979.

[McCracken 1979]
McCracken, N.J., An investigation of a programming language with a polymorphic type structure. Ph.D. dissertation, Syracuse University, 1979.

[McCracken 1980]
McCracken, N.J., Private correspondence, 1980.

[Milner 1978]
Milner, R., A theory of type polymorphism in programming. University of Edinburgh, 1978, revised.

[Oles 1982]
Oles, Frank, A category-theoretic approach to the semantics of programming languages. Ph.D. dissertation, Syracuse University, 1982.

[Reynolds 1974]
Reynolds, J.C., Towards a theory of type structure. Colloquium of Programming, Paris, 1974.

[Reynolds 1978]
Reynolds, J.C., Syntactic control of interference. Fifth Annual Symposium on Principles of Programming Languages, January 1978.

[Scott 1980]
Scott, Dana, A space of retracts. Manuscript, Merton College, Oxford, April 1980.

[Stenlund 1972]
Stenlund, Soren, Combinators, lambda terms, and proof theory. D. Reidal, Dordrecht, Holland, 1972.

Appendix

This appendix contains the proofs of all the material of Sections III and IV, including a statement of definitions and of supporting lemmas omitted from the text of the paper. Most of the proofs are taken from [Scott 1980]. However, although the main ideas are from Scott, the amount of detail and wording may be different in this presentation. Therefore, the author takes full responsibility for any errors or omissions.

Proofs for Section III

Definition: An element $e \in D$ is finite iff for any subset $X \subseteq D$, if $e \leq \cup X$ then there exists a finite set $Xf \subseteq X$ such that $e \leq \cup Xf$.

Notation: The set of finite elements of D is denoted by E.

Definition: A subset $X \subseteq D$ is consistent iff every finite subset has an ub in D.

Definition: A subset $X \subseteq D$ is directed iff every finite subset has an ub in X, (or equivalently, if X is nonempty and every $x, y \in X$ has an ub in X).

3.1 Lemma: If e is a finite element of a poset D and $X \subseteq D$ is a directed set, then if $e \leq \cup X$, $\exists x \in X$ such that $e \leq x$.

Proof: If $e \leq \cup X$, then there is a finite set $Xf \subseteq X$ such that $e \leq \cup Xf$. But if X is directed, $\cup Xf$ is an element of X. □

Definition: A nonempty poset D is a finitary domain iff

i) every consistent subset has a lub and

ii) every element is the lub of some finite elements.

As an alternative to the first condition, we have:

3.2 Proposition: Every consistent subset of a poset D has a lub iff

ia) every directed subset has a lub and

ib) every bounded subset has a lub.

Proof:

(=>) Directed subsets and bounded subsets are consistent.

(<=) Let X be a consistent subset of D. Show $\sqcup X$ exists in D. Every finite subset of X is bounded, so it has a lub. Consider the union of X with the lub's of finite subsets of X. This is a directed set, so it has a lub, which is also an ub of X. Let z be this ub of X. Suppose y is another ub of X and $y \leq z$. But if y is an ub of X, then $y \geq$ the lub of every finite subset of X (and $y \geq$ every element of X). So y is also an ub of the union of X with the lub's of all finite subsets of X, so that $y \geq z$, the lub of this set. Therefore z is also the lub of X.

3.3 Proposition: A lub of a finite set of finite elements in a finitary domain D is also finite.

Proof: Let X be a finite set of finite elements such that $\sqcup X$ exists. To show that $\sqcup X$ is a finite element, we assume that there is some set $Y \subseteq D$ such that $\sqcup X \leq \sqcup Y$. We must show that there exists a finite set $Yf \subseteq Y$ such that $\sqcup X \leq \sqcup Yf$.

If $\sqcup X \leq \sqcup Y$, then for all x in X, $x \leq \sqcup Y$. The elements of X *are* finite, so for each x, $\exists$ a finite set $Yx \subseteq Y$ such that $x \leq \sqcup Yx$.

Let Yf be the union of all the Yx's. This set is finite and is contained in Y. $\sqcup Yf$ exists since Yf is a bounded set. And, $\forall x \in X$, $x \leq \sqcup Yf$, so $\sqcup X \leq \sqcup Yf$.

□

Notation: From now on, D and D' will denote finitary domains, unless otherwise stated.

Definition: A continuous function from D to D' is a function f such that

$$\text{for all directed sets } S \subseteq D,\ f(\sqcup S) = \sqcup f(S).$$

3.4 Proposition: A function f from D to D' is continuous iff

$$\forall e' \in E',\ x \in D:\quad e' \leq f(x) \text{ iff } \exists e \in E.\ e \leq x \text{ and } e' \leq f(e).$$

Proof: Note that f continuous implies f monotone. (Since $x \leq y$ implies that $\{x, y\}$ is a directed set.)

(=>) Assume that $\forall$ directed sets $S \subseteq D$, $f(\cup S) = \cup f(S)$. Let $e' \in E$, $x \in D$:

a.(=>) Assume $e' \leq f(x)$. Show $\exists e \in E.\ e \leq x$ and $e' \leq f(e)$.

Now D is finitary, so $x = \cup\{e \in E \mid e \leq x\}$. f is continuous and the set $\{e \in E \mid e \leq x\}$ is directed, so $f(x) = \cup\{f(e) \mid e \in E \text{ and } e \leq x\}$. Then since e' is finite in D' and $e' \leq \cup\{f(e) \mid e \in E \text{ and } e \leq x\}$ then $e' \leq f(e)$ for some e such that $e \leq x$.

b.(<=) Assume $\exists e \in E.\ e \leq x$ and $e' \leq f(e)$. Show $e' \leq f(x)$.

f is monotone so $e \leq x$ implies $f(e) \leq f(x)$. So $e' \leq f(e) \leq f(x)$.

(<=) Assume that $\forall e' \in E'$, $x \in D$, $e' \leq f(x)$ iff $\exists e \in E.\ e \leq x$ and $e' \leq f(e)$. Show f is continuous.

a. Show f is monotonic.

Let $y, z \in D$ be such that $y \leq z$. Now D' is finitary (algebraic) so $f(y) = \cup\{e' \in E' \mid e' \leq f(y)\}$ and $f(z) = \cup\{e' \in E' \mid e' \leq f(z)\}$. To show $f(y) \leq f(z)$, we show $\{e' \in E' \mid e' \leq f(y)\} \subseteq \{e' \in E' \mid e' \leq f(z)\}$. So we must show that $\forall e' \in E'$, if $e' \leq f(y)$, then $e' \leq f(z)$.

So let $e' \in E'$ be such that $e' \leq f(y)$. Then there is an e in E such that $e \leq y$ and $e' \leq f(e)$. Now $e \leq y \leq z$, so we know that $\exists e \in E.\ e \leq z$ and $e' \leq f(e)$. Then we can use our original assumption to conclude that $e' \leq f(z)$.

b. To show f is continuous, let $S \subseteq D$ be a directed set and show $f(\cup S) \leq \cup f(S)$. Now D is algebraic, so $f(\cup S) = \cup\{e' \in E' \mid e' \leq f(\cup S)\}$. To show $\cup\{e' \in E' \mid e' \leq f(\cup S)\} \leq \cup f(S)$, show that every element e' in the first set approximates some element of the second set, $f(S)$. Let $e' \in E'$ be such that $e' \leq f(\cup S)$. Now $\exists e \in E.\ e \leq \cup S$ and $e' \leq f(e)$, by our original assumption. Now e is finite and approximates the lub of a directed set, so $\exists\ x \in S$ such that $e \leq x$. Now we know that $\exists e \in E.\ e \leq x$ and $e' \leq f(e)$, so we can conclude that $e' \leq f(x)$, for some $x \in S$. □

<u>3.5 Proposition:</u> For any set of functions $F \subseteq D \to D'$, whenever, $\forall x \in D$, $\cup\{f(x) \mid f \in F\}$

exists, then $\sqcup F$ exists as a continuous function and is defined by $\sqcup F(x) = \sqcup\{f(x) \mid f\in F\}$, $\forall x\in D$.

Proof: Let g be the function such that $\forall\, x\in D$, $g(x) = \sqcup\{f(x) \mid f\in F\}$. If g exists in $D\to D'$, it is already continuous. Therefore, we must show that g is the lub of F.

1. g is an ub of F, since $\forall f\in F$, $f \leq g$, (since $\forall x\in D$, $f(x)\leq g(x)$).

2. Suppose there is another ub of F, say h, and $h \leq g$. But if h is an ub of F, $\forall f\in F$, $\forall x\in D$, $f(x) \leq h(x)$. But then $\forall x\in D$, $\sqcup\{f(x) \mid f\in F\} \leq h(x)$. So g is the lub of F. □

In order to show that $D\to D'$ is a finitary domain whenever D and D' are, we introduce "step functions" whose finite lub's will be the finite elements of $D\to D'$, whenever they exist.

Definition: A step function $[e,e']:D\to D'$ is defined by:

$$\forall x\in D,\quad [e,e'](x) = e' \quad \text{if } e\leq x$$
$$= \bot' \quad \text{otherwise.}$$

This is a continuous function with a two-point image in D'.

3.6 Lemma: Let f be a continuous function from D to D'. Then

(1) $[e,e'] \leq f$ iff $e' \leq f(e)$

(2) $f = \sqcup\{[e,e'] \mid e'\leq f(e)\}$

and (3) The $[e,e']$ are all finite in $D\to D'$.

Proof:

(1)(=>) Let$[e,e'] \leq f$. Now $[e,e'](e) = e'$ by the definition of a step function. Then, since $[e,e'](e) \leq f(e)$, $e' \leq f(e)$.

(<=) Let $e'\leq f(e)$. Let $x\in D$. Show $[e,e'](x)\leq f(x)$.

Either $e\leq x$ and $[e,e'](x) = e' \leq f(e) \leq f(x)$ or

$e\not\leq x$ and $[e,e'](x)=\bot' \leq f(x)$.

(2) a. That $\sqcup\{[e,e'] \mid e'\leq f(e)\} \leq f$ is obvious from part (1).

b. Show that $f \leq \sqcup\{[e,e'] \mid e'\leq f(e)\}$.

Since D' is finitary, $\forall x\in D$, $f(x) = \sqcup\{e'\in E' \mid e'\leq f(x)\}$. By lemma 3.4, we then have $f(x) = \sqcup\{e'\in E' \mid \exists e\in E.\ e\leq x \text{ and } e'\leq f(e)\}$. Then for each e' in this set (i.e. $\exists e\in E.\ e\leq x$ and $e'\leq f(e)$), $\exists e\in E$ such that $[e,e'](x) = e'$.

So $\{e'\in E' \mid \exists e\in E.\ e\leq x \text{ and } e'\leq f(e)\} \subseteq \{[e,e'](x) \mid e'\leq f(e)\}$. Then $f \leq \sqcup\{[e,e'] \mid e'\leq f(e)\}$.

(3) Suppose that $[e,e']\leq\sqcup F$ for some set $F\subseteq D\to D'$.

By part (1), $e'\leq \sqcup F(e) = \sqcup\{f(e) \mid f\in F\}$, by the definition of $\sqcup F$. Now e' is finite in D', so there is some finite set $Ff \subseteq F$ such that $e' \leq \sqcup\{f(e) \mid f\in Ff\}$. Now Ff is a bounded set, so its lub exists, and $e' \leq \sqcup Ff(e)$. Then by part (1), $[e,e'] \leq \sqcup Ff$, so that $[e,e']$ is finite in $D\to D'$.

Parts (2) and (3) of this lemma show that the step functions are sufficient to establish the algebraic nature of $D\to D'$. In addition, we can see that, since finite lub's of finite elements are finite,

$E\to E' = \{[e1,e1']\sqcup\ldots\sqcup[en,en'] \mid$ the lub exists for all $e1,\ldots,en\in E$ and for all $e1',\ldots,en'\in E'\}$.

3.7 Theorem: If D and D' are finitary domains, then $D\to D'$ is also.

Proof

(i) Let $F \subseteq D\to D'$ be a consistent set of functions.

If $\sqcup F$ does exist, then $\forall x\in D$, $\sqcup F(x)$ will be $\sqcup\{f(x) \mid f\in F\}$. Now consider any finite subset of $\{f(x) \mid f\in F\}$, say $\{f(x) \mid f\in G\}$, where G is a finite subset of F. But an ub of G exists since F is consistent in $D\to D'$, so an ub of $\{f(x) \mid f\in G\}$ exists and $\{f(x) \mid f\in F\}$ is consistent in D'. Now consistent sets in D' have lubs, so $\sqcup\{f(x) \mid f\in F\}$ exists in D' for all $x\in D$, so $\sqcup F$ exists in $D\to D'$. □

(ii) That every functions f is the lub of finite elements is shown by lemma 3.6, part (2). □

Definition: Categorical product

For every pair of objects D and D', there is an object $D\times D'$ and morphisms $p:D\times D'\rightarrow D$ and $p':D\times D'\rightarrow D'$ such that for any object C and morphisms $f:C\rightarrow D$ and $f':C\rightarrow D'$, $\exists$ a unique morphism $\langle f,f'\rangle$ such that the following diagram commutes:

$$\begin{tikzcd}
 & C \arrow[dl, "f"'] \arrow[d, "{\langle f,f'\rangle}"] \arrow[dr, "f'"] & \\
D & D\times D' \arrow[l, "p"] \arrow[r, "p'"'] & D'
\end{tikzcd}$$

Now in our case, we can define $D\times D'$ to be the usual pairs of elements:

Definition: $D\times D'=\{\langle x,x'\rangle \mid x\in D, x'\in D'\}$, where $\langle x,x'\rangle\leq\langle y,y'\rangle$ iff $x\leq x'$ and $y\leq y'$.

3.8 Lemma: Let $Z \subseteq D\times D'$. Let $X = \{x \mid \exists x'.\langle x,x'\rangle\in Z\}$ and $X' = \{x' \mid \exists x.\langle x,x'\rangle\in Z\}$. Then

(1) $\langle z,z'\rangle = \cup Z$ iff $z = \cup X$ and $z' = \cup X'$.

(2) Z is a consistent (directed) set iff X and X' are consistent (directed) sets.

Proof: (1) This is obvious from the definition of $\leq$ in $D\times D'$.

(2) a. Let Z be a consistent set.

Let Xf be a finite subset of X and x' be an element of X'. Consider the set $\{\langle x,x'\rangle\in Z \mid x\in Xf\}$. This set is a finite subset of Z and Z is consistent in $D\times D'$, so it has an ub, say $\langle z,z'\rangle$. Then $\forall x\in Xf$, $x \leq z$, so z is an ub of Xf, so X is consistent in D.

A similar argument shows that X' is consistent in D'.

b. Let X and X' be consistent sets.

Let Zf be a finite subset of Z. Let $Xf = \{x \mid \exists x'.\langle x,x'\rangle\in Zf\}$. This set is a finite subset of X, which is consistent in D, so it has an ub, say z. Now let $Xf' = \{x' \mid \exists x.\langle x,x'\rangle\in Zf\}$. By similar reasoning, it has an ub in D', say z'. Then $\langle z,z'\rangle$ is an ub of Zf, so Z is consistent in $D\times D'$.

The argument for directed sets is similar. $\square$

<u>3.9 Proposition:</u> If D and D' are finitary domains, then so is $D\times D'$, where the set of finite elements is $E\times E'$.

<u>Proof:</u>

A. The elements of $E\times E'$ are finite elements in $D\times D'$.

Let $\langle e,e'\rangle\in E\times E'$. Suppose $\langle e,e'\rangle \leq \cup Z = \langle \cup X,\cup X'\rangle$, for some set $Z \subseteq D\times D'$. Then $e\leq\cup X$, $e'\leq\cup X'$ and e,e' are finite in D and D', so there exist finite sets $Xf \subseteq X$ and $Xf' \subseteq X'$ such that $e \leq \cup Xf$ and $e' \leq \cup Xf'$. But then $\{\langle x,x'\rangle | x\in Xf,\ x'\in Xf'\}$ is a finite subset of Z and its lub is $\langle \cup Xf,\cup Xf'\rangle$ and $\langle e,e'\rangle \leq \langle \cup Xf,\cup Xf'\rangle$. □

B. Every element of $D\times D'$ is the lub of finite elements.

Let $\langle x,x'\rangle\in D\times D'$. Now D and D' are finitary, so $x=\cup\{e\in E | e\leq x\}$and $x'=\cup\{e'\in E' | e'\leq x'\}$.

Then $\langle x,x'\rangle=\cup\{\langle e,e'\rangle | e\leq x, e'\leq x'\}$. □

C. That all consistent sets have lubs is obvious from lemma 3.8.

<u>3.10 Theorem:</u> Let $\langle x,x'\rangle \in D\times D'$. Define $p(\langle x,x'\rangle)=x$ and $p'(\langle x,x'\rangle)=x'$. If C is any other finitary domain, and $f:C\rightarrow D$ and $f':C\rightarrow D'$ are continuous functions, define $\forall x\in C$, $\langle f,f'\rangle(x)=\langle f(x),f'(x)\rangle$. Then $D\times D'$ is the categorical product.

<u>Proof:</u>

A. That p and p' are continuous is immediate from lemma 3.8.

B. Show that $\langle f,f'\rangle$ is continuous.

Let X be a directed set in C. Then $\langle f,f'\rangle(\cup X)=\langle f(\cup X),\ f'(\cup X)\rangle$

$=\langle \cup f(X),\cup f(X)'\rangle$, f&f' are continuous.

$= \cup\langle f(X),f'(X)\rangle$, Lemma 3.8.

$= \cup\langle f,f'\rangle(X)$.

C. Show that the diagram commutes:

Show $f(x)=p\circ(\langle f,f'\rangle)\ (x)$ (and similarly for f',p').

$p(\langle f,f'\rangle(x)) = p(\langle f(x),f'(x)\rangle) = f(x)$. □

D. Show that $\langle f,f'\rangle$ is unique:

From the calculations in part C., we can see that the first component of $\langle f,f'\rangle(x)$ must be $f(x)$, and the second must be $f'(x)$, so that $\langle f,f'\rangle$ is unique. □

<u>Definition:</u> A category is cartesian closed iff it has finite products and for every pair of objects D and D', there is an object D=>D' and a morphism $ap:(D{=>}D')\times D\to D'$ such that for every object C and morphism $f:C\times D\to D'$, $\exists$ a unique morphism $h:C\to(D{=>}D')$ such that the following diagram commutes:

$$\begin{tikzcd}
C\times D \arrow[r, "h\times I"] \arrow[dr, "f"'] & (D{=>}D')\times D \arrow[d, "ap"] \\
 & D'
\end{tikzcd}$$

The morphism h is usually defined as $ab(f)$ for a function $ab:(C\times D\to D')\to(C\to(D{=>}D'))$.

<u>3.11 Theorem:</u> Let C, D, and D' be finitary domains.

Let $f \in D{=>}D'$ and $x \in D$. Define $ap(\langle f,x\rangle) = f(x)$.

Let $f \in C\times D\to D'$, $x\in C$, and $y\in D$. Define $ab(f)(x)(y) = f(\langle x,y\rangle)$.

Then the finitary domains and their continuous functions form a cartesion closed category.

<u>Proof:</u>

1. We have already shown that all finite products exist.

2. Show that ap is a continuous function. Let $Z \subseteq (D{=>}D')\times D$ be a directed set.

Then

$ap(\cup Z) = ap(\cup\{\langle f,a\rangle \mid \langle f,a\rangle\in Z\})$, where $f\in D{=>}D'$ and $a\in D$,

$= ap(\langle\cup\{f \mid \langle f,a\rangle\in Z\},\ \cup\{a' \mid \langle f',a'\rangle\in Z\}\rangle)$, lub's distribute over products,

$= \cup\{f \mid \langle f,a\rangle\in Z\}(\cup\{a' \mid \langle f',a'\rangle\in Z\})$, by the definition of ap,

$= \cup\{f(\cup\{a' \mid \langle f',a'\rangle\in Z\}) \mid \langle f,a\rangle\in Z\}$, by the def. of lub's of functions,

$= \cup\{\cup\{f(a') \mid \langle f',a'\rangle \in Z\} \mid \langle f,a\rangle \in Z\}$, since each f is continuous,

$= \cup\{f(a') \mid \langle f,a\rangle \in Z \text{ and } \langle f',a'\rangle \in Z\}$.

Now the set $\{f(a) \mid \langle f,a\rangle \in Z\}$ is contained in the last set above, so $\cup\{f(a') \mid \langle f,a\rangle \in Z \text{ and } \langle f',a'\rangle \in Z\} \geq \cup\{f(a) \mid \langle f,a\rangle \in Z\}$. On the other hand, Z is directed so that for all $\langle f,a\rangle$ in Z and $\langle f',a'\rangle$ in Z, there is an element $\langle f'',a''\rangle$ in Z that is an ub of $\langle f,a\rangle$ and $\langle f',a'\rangle$. Application is monotonic, so $\cup\{f(a') \mid \langle f,a\rangle \in Z \text{ and } \langle f',a'\rangle \in Z\} \leq \cup\{f(a) \mid \langle f,a\rangle \in Z\}$. Then the last equation above is

$= \cup\{f(a) \mid \langle f,a\rangle \in Z\}$

$= \cup\{ap(\langle f,a\rangle \mid \langle f,a\rangle \in Z\}$. $\sqcup$

3. Show that if $f \in C\times D \to D'$ is continuous, then ab(f) is continuous. (This shows that h is a morphism in the category.)

   Let $X \subseteq C$ be a directed set. Let $y \in D$. Then

   $ab(f)(\cup X)(y) = f(\langle \cup X, y\rangle)$

   $= f(\cup\{\langle x,y\rangle \mid x \in X\})$, since X is not empty,

   $= \cup\{f(\langle x,y\rangle) \mid x \in X\}$, since f is continuous,

   $= \cup\{ab(f)(x)(y) \mid x \in X\}$

   $= \cup\{ab(f)(x) \mid x \in X\}(y)$. $\sqcup$

4. Show that if $f \in C\times D \to D'$ is continuous, then for all x in C, ab(f)(x) is continuous. (This shows that h(x) is a morphism in D=>D'.)

   Let $Y \subseteq D$ be a directed set. Then

   $ab(f)(x)(\cup Y) = f(\langle x, \cup Y\rangle)$

   $= f(\cup\{\langle x,y\rangle \mid y \in Y\})$, since Y is not empty,

   $= \cup\{f(\langle x,y\rangle) \mid y \in Y\}$, since f is continuous,

   $= \cup\{ab(f)(x)(y) \mid y \in Y\}$. $\sqcup$

5. Show that the diagram commutes.

   Let $\langle x,y\rangle \in C\times D$. Show that $f(\langle x,y\rangle) = ap((h\times I)(\langle x,y\rangle))$:

   $ap((n\times I)(\langle x,y\rangle)) = ap((ab(f)\times I)(\langle x,y\rangle)$

$$= ap(\langle ab(f)(x),y\rangle)$$

$$= ab(f)(x)(y)$$

$$= f(\langle x,y\rangle). \quad \square$$

6. The above equations show that h is the unique function that makes the diagram commute.

Proofs for Section IV

Definition: For finitary domains D and D', D is a retract of D' iff there exist continuous functions $i:D \to D'$ and $j:D' \to D$ such that $j \circ i = I$ on D (the identity function on D).

Definition: A retraction on a finitary domain D is a continuous function $r:D \to D$ such that $r \circ r = r$.

4.1 Lemma: $r(D) = Fix(r)$, where $Fix(r) \equiv \{x \in D \mid r(x)=x\}$.

Proof: If $x \in Fix(r)$, then obviously x is in the image of r.

If $x \in r(D)$, then there is some $y \in D$ such that $r(y) = x$. Then $r(x) = r(r(y)) = r(y) = x$. □

4.2 Proposition: If the image of a retraction r on D is a finitary domain, then it is a retract of D.

Proof: Consider the continuous functions $i:r(D) \to D$, where $\forall x \in r(D)$, $i(x)=x$, and $r':D \to r(D)$, where $r'=r \restriction r(D)$. Then for all $x \in r(D)$, $(i \circ r')(x) = i(r'(x)) = r'(x) = r(x) = x$. □

4.3 Lemma: r(D) has lub's of all consistent subsets. Furthermore, r(D) is directed complete, i.e. if X is a directed set, $X \subseteq r(D)$, then $\sqcup X$ in r(D) = $\sqcup X$ in D.

Proof:

i) Show that r(D) has lub's of all directed sets and is directed complete.

Let $X \subseteq r(D)$ be a directed set. Show that $\sqcup X$ in D is an element of r(D) by showing $r(\sqcup X) = \sqcup X$.

$r(\sqcup X) = \sqcup \{r(x) \mid x \in X\}$, since r is continuous,

$= \sqcup \{x \mid x \in X\}$, since $X \subseteq r(D)$ implies $r(x)=x$, $\forall x \in X$.

$= \sqcup X$. □

ii) Show that r(D) has lub's of all bounded sets.

Let $X \subseteq r(D)$ be a bounded set. D is finitary, so $\sqcup X$ exists in D. Let y =

$r(\sqcup X)$, which is an element of $r(D)$, and $\forall x \in X$, $x = r(x) \leq y$, so y is an ub of X in $r(D)$. Suppose there is some other ub of X in $r(D)$, say z. Then $\forall x \in X$, $x \leq z$. Then z is also an ub of X in D, where $\sqcup X$ is least. Since r is monotonic, $r(\sqcup X) \leq r(z)$. Then $y = r(\sqcup X)$ and $z = r(z)$, so $y \leq z$ and must be the lub of X. □

Definition: A retraction of a finitary domain D is a finite retraction iff it is a finite element of $D \to D$.

Note that a finite retraction also must be finitary, since the image of a finite function is always finite.

Definition: A continuous function f is a preretraction iff $f \leq f \circ f$.

4.4 Lemma:

a. If f is a finite preretraction, then the least retraction that it approximates, denoted ∇f, is also finite.

b. The finite retractions appromimating a given retraction form a directed set.

Proof: Let D be an arbitrary finitary domain.

a. Let $f: D \to D$ be a finite preretraction, i.e. $f \leq f^2 = f \circ f$. Then by monotonicity, $f \leq f^2 \leq f^3 \ldots. \leq f^n \leq \ldots.$.

Does this sequence go on forever? Recall that $f = [e1, e1'] \sqcup \ldots \sqcup [en, en']$ for some $e1, \ldots, en \in D$, $e1', \ldots, en' \in D'$, and that since f is finite all the lub's of the ei's exist. Let $F \subseteq D$ be the finite poset generated as all lub's of $\{e1', \ldots, en'\}$. (F is finitary since it is finite in size and all lub's exist.) Let $j: D \to D$ be such that $\forall x \in D$, $j(x) = \sqcup \{e \in F \mid e \leq x\}$. Now j is a finitary retraction of D since $j(D) = F$. Now $j \circ f = f$ since j is a retraction and $f(D) \subseteq j(D)$. Now $(f \circ j)$ is a pretraction of F:

Let $e \in F$. $f(j(e)) \leq f(f(j(e)))$, since f is a preretraction on D.

Then $f(j(e)) = f(j(f(j(e)))$, since $f = j \circ f$.

Also, $f \circ j \leq f^2 \circ j \leq f^3 \circ j \leq \ldots.$in $F \to F$. Now $F \to F$ is finite in size, since F is, so this sequence cannot strictly increase forever. Let n be first integer such that $f^n \circ j = f^{n+1} \circ j$. But then $f^n \circ j \circ f = f^{n+1} \circ j \circ f$, so $f^{n+1} = f^{n+2}$ (using $j \circ f = f$). Similarly, $f^{n+2} = f^{n+3}, \ldots$ and finally $f^{n+1} = f^{n+1} \circ f^{n+1}$. So f^{n+1} is a retraction on D.

Now let r be any retraction such that $f \leq r$. Then $\forall m$, $f^m \leq r$, ($f^2 \leq r^2 = r$, etc.). So f^{n+1} is the least retraction containing f. We will henceforth denote it by ∇f. Furthermore, ∇f is finite since the composition of finite functions is finite.

b. Let $r: D \rightarrow D$ be any retraction. Let f and g be finite retractions such that $f \leq r$ and $g \leq r$. Now $f \sqcup g$ exists since $\{f, g\}$ is bounded by r, and $f \sqcup g \leq r$ and $f \sqcup g$ is finite. Then $f = f \circ f \leq (f \sqcup g) \circ (f \sqcup g)$ and $g = g \circ g \leq (f \sqcup g) \circ (f \sqcup g)$, so $f \sqcup g \leq (f \sqcup g) \circ (f \sqcup g)$. Thus $f \sqcup g$ is a preretraction, and by part a there is a least finite retraction $\nabla(f \sqcup g)$ such that $f \sqcup g \leq \nabla(f \sqcup g)$. So $\nabla f \sqcup g \leq r$. Thus any two finite retractions approximating r have an ub approximating r, so that the set of finite retractions approximating r is directed. □

4.5 Theorem: Let $r: D \rightarrow D$ be a retraction. Then the following are equivalent:

i) r is finitary.

ii) $r(x) = \sqcup\{r(e) \mid e \in E,\ e \leq r(x) \text{ and } e \leq r(e)\}$.

(Note that this lub is the same in r(D) as in D since r(D) is directed complete.)

iii) r is the lub of some finite retractions.

Proof: In this proof, we will write $\sqcup[D]$ and $\sqcup[r(D)]$ to distinguish the lub's in D and r(D), and E[D] and E[r(D)] to distinguish their finite elements.

(i=>ii) Assume r is finitary.

Lemma: Let $X = \{r(e) \mid e \leq r(e),\ e \in E[D]\}$. Show $X = E[r(D)]$, the finite elements of r(D).

a. Let $d \in E[r(D)]$. D is finitary so $d = \sqcup[D]\{e \in E[D] \mid e \leq d\}$. Then $d = r(d) = \sqcup[r(D)]\{r(e) \mid e \leq d, e \in E[D]\}$, since r is continuous, r(D) is directed complete, and $\{e \in E[D] \mid e \leq d\}$ is a directed set. Now d is finite in $r(D)$ and and d approximates the lub of a directed set, so there is an $e \in E[D]$ such that $d \leq r(e)$. But d is also equal to the lub of the set, so $r(e) \leq d$, thus there is an $e \in D$ such that $d = r(e)$. Now $e \leq d$ so $e \leq r(e)$, so d is in the set X.

b. Let $r(e)$ be such that $e \leq r(e)$ and $e \in E[D]$, i.e. $r(e) \in X$. Show $r(e) \in E[r(D)]$. Let S be any set contained in $r(D)$ such that $r(e) \leq \cup[r(D)]S$. We must show that there is a finite set $Sf \subseteq S$ such that $r(e) \leq \cup[r(D)]Sf$. Now $\cup[D]S$ exists since $\cup[r(D)]S$ is an ub on S in D, and bounded sets have lub's. Also we must have $e \leq r(e) \leq \cup[D]S \leq \cup[r(D)]S$, since if <u>not</u> $r(e) \leq \cup[D]S$, then $r(e)$ would be a smaller ub on S in $r(D)$. Now e is finite in D and $e \leq \cup[D]S$ so there is a finite set $Sf \subseteq S$ such that $e \leq \cup[D]Sf$. Then $r(e) \leq r(\cup[D]Sf)$, by monotonicity. But $\cup[d]Sf \leq \cup[r(D)]Sf$ and $r(\cup[r(D)]Sf) = \cup[r(D)]Sf$, so we have $r(e) \leq \cup[r(D)]Sf$. □

Now if r is finitary, $\forall x \in D$, $r(x) \in r(D)$ implies $r(x) = \cup\{d \mid d \leq r(x) \text{ and } d \in E[r(D)]\} = \cup\{r(e) \mid e \in E[D], e \leq r(x) \text{ and } e \leq r(e)\}$, by the above lemma. □

(ii=>i) Assume $\forall x \in D$, $r(x) = \cup\{r(e) \mid e \in E[D], e \leq r(x) \text{ and } e \leq r(e)\}$. Show r is finitary. For <u>all</u> retractions, $r(D)$ has lub's of consistent sets. Now every element in $r(D)$ is $r(x)$ for some $x \in D$. We showed in part b. of the lemma above that the elements $r(e)$ such that $e \leq r(e)$ and $e \in E[D]$ are all finite in $r(D)$, so the above equation shows that every element of $r(D)$ is the lub of some finite elements in $r(D)$.

(ii=>iii) Assume $\forall x \in D$, $r(x) = \cup\{r(e) \mid e \in E[D], e \leq r(x) \text{ and } e \leq r(e)\}$.

Show r is the lub of finite retractions.

(*) For any continuous function: $r = \cup[D \to D]\{[e, e'] \mid e' \leq r(e)\}$, since $D \to D$ is finitary. To show that r is the lub of some finite retractions, we just show that for every finite function in $\{[e, e'] \mid e' \leq r(e)\}$ there is a finite retraction f such that $[e,e']\leq f\leq r$. So let $[e,e']$ be any finite function such that $e'\leq r(e)$ and construct f as follows:

Consider equation ii for $r(e) = \cup\{r(\underline{e})\mid\underline{e}\in E,\ \underline{e}\leq r(e) \text{ and } \underline{e}\leq r(\underline{e})\}$. Now $e'\leq r(e)$, which is the lub of a directed set in D and e' is a finite element in D, so for some $r(e'')$ in the set, $e'\leq r(e'')$, and then also $e''\leq r(e)$ and $e''\leq r(e'')$ by definition of the set. Now e' and e'' have an ub $r(e)$, so they have a lub: $e'\cup e''$, and we have $e'\cup e''\leq r(e'\cup e'')$. Then by equation *: $[e'\cup e'',\ e'\cup e'']\leq r$ and also $[e,e'\cup e'']\leq r$, since $e'\cup e''\leq r(e)$. Let $f=[e,e'\cup e'']\cup[e'\cup e'',e'\cup e'']$. This always exists since both are bounded by r, and then $f\leq r$. Now f is a finite function, and we need to show that f is a retraction.

Now f is the function:

```
$\lambda x.$if $e\leq x$ but $e'\cup e''\not\leq x$ then $e'\cup e''$
    else if $e\not\leq x$ but $e'\cup e''\leq x$ then $e'\cup e''$
  else if $e\cup(e'\cup e'')\leq x$ then $e'\cup e''$ else $\perp$,
```

which is just: $\lambda x.$if $e\leq x$ or $e'\cup e''\leq x$ then $e'\cup e''$ else $\perp$.

Now if e or $e'\cup e''$ is $\perp$, then f is the constant function $e'\cup e''$. Otherwise, f has a two point image with $f(\perp)=\perp$ and $f(e'\cup e'')=e'\cup e''$. In either case, f is a retraction. □

(iii=>ii) Assume r is the lub of finite retractions.

Then we can write $r=\cup\{f\in E\to E\mid f\leq r \text{ and } f=f\circ f\}$. By lemma 4.4, this is a directed set, so

$$\forall x\in D:\quad r(x) = \cup\{f(x)\mid f\leq r \text{ and } f=f\circ f,\ f\in E\to E\}$$

$$= r(r(x)) = \cup\{r(f(x))\mid f\leq r \text{ and } f=f\circ f,\ f\in E\to E\},$$ since r is continuous.

Since each f is finite in $D\to D$, each $f(x)$ is finite in D. Now each $f(x)\leq r(x)$ and $f(x)\leq r(f(x))$, so each $f(x)$ is a finite element in equation ii. □

Definition: Let $\rho:(D\to D)\to(D\to D)$ be defined as the function such that $\forall g\in D\to D$:

$$\rho(g)=\sqcup\{\nabla f\mid f\in E\to E. f\leq g \text{ and } f\leq f\circ f\}.$$

4.6 Lemma: If $r\in \mathrm{FRet}(D)$, then $\rho(r)=r$.

Proof: If r is finitary, then $r=\sqcup\{f\in E\to E\mid f\leq r \text{ and } f=f\circ f\}$ by the theorem 4.5, part iii. Now show $\{f\in E\to E\mid f\leq r \text{ and } f=f\circ f\} = \{\nabla f\mid f\leq r \text{ and } f\leq f\circ f,\ f\in E\to E\}$. Let f be in the lhs set. But for a finite retraction $f=\nabla f$ so f is in the rhs set. Let ∇f be from the rhs set. Now ∇f is the least retraction that f approximates and is finite, so ∇f is in the lhs set.

So $r = \sqcup\{\nabla f\mid f\leq r \text{ and } f\leq f\circ f,\ f\in E\to E\} = \rho(r)$. □

4.7 Theorem: FRet(D) is a finitary domain with the finite retractions as its finite elements, and FRet(D) is a retract of $D\to D$ by the function ρ.

Proof:

a. Show that ρ is a retraction.

Now ρ is obviusly continous. Let $r\in D\to D$.

$\rho(\rho(r)) = \rho(\sqcup\{\nabla f\mid f\leq r \text{ and } f\leq f\circ f,\ f\in E\to E\})$

$=\sqcup\{\rho(\nabla f)\mid f\leq r \text{ and } f\leq f\circ f,\ f\in E\to E\}$, since ρ is continuous.

But ∇f is finite and thus finitary, so this equation is

$= \sqcup\{\nabla f\mid f\leq r \text{ and } f\leq f\circ f,\ f\in E\to E\}$, by lemma 4.6

$= \rho(r)$.

b. Show that $\mathrm{FRet}(D)=\rho(D\to D)$.

1. $\mathrm{FRet}(D)\subseteq \rho(D\to D)$, by lemma 4.6 since $r\in \mathrm{FRet}(D)\Rightarrow\rho(r)=r$.

2. Show that $\rho(D\to D)\subseteq \mathrm{FRet}(D)$. Let $r\in D\to D$. Show $\rho(r)$ is a finitary retraction on D.

i. Show that $\rho(r)$ is a retraction. Since $\{\nabla f\mid f\leq r \text{ and } f\leq f\circ f,\ f\in E\to E\}$ is a directed set and $\circ$ is continuous: $\rho(r)\circ\rho(r)=\sqcup\{\nabla f\circ\nabla f\mid f\leq r \text{ and } f\leq f\circ f,\ f\in E\to E\}$. Then each $\nabla f=\nabla f\circ\nabla f$, so $\rho(r)\circ\rho(r)=\rho(r)$. □

ii. Each ∇f is a finite retraction, so by theorem 4.5, $\rho(r)$ is the lub of some finite retractions. Therefore $\rho(r)$ is finitary.

c. Show that FRet(D) is finitary, i.e. ρ is a finitary retraction on D→D.

1. Fret(D) has lub's of consistent sets since it is the range of a retraction.

2. Show that every element of FRet(D)(=ρ(D→D)) is the lub of finite elements of FRet(D).

i. Show that if f is a finitary retraction that is a finite element of D→D, then it is also a finite element of FRet(D). Let F⊆ FRet(D) be a set of functions such that $f \leq \sqcup$[FRet(D)]F. Then $\sqcup$[FRet(D)F = $\sqcup$[D→D]F, since FRet(D) is the codomain of a retraction, ρ. Then f is finite in D→D so it is also finite in FRet(D). □

ii. Let r∈FRet(D). Then r is a finitary retract on D, so it is the lub of finite retractions by theorem 4.5. But these finite retractions are finite elements of FRet(D). □