E. POLL
C. HEMERIK
H. M. M. TEN EIKELDER

CPO-models for second order lambda calculus with recursive types and subtyping

Informatique théorique et applications, tome 27, nº 3 (1993), p. 221-260.
<http://www.numdam.org/item?id=ITA_1993__27_3_221_0>

Informatique théorique et Applications/Theoretical Informatics and Applications
(vol. 27, n° 3, 1993, p. 221 à 260)

CPO-MODELS FOR SECOND ORDER LAMBDA CALCULUS WITH RECURSIVE TYPES AND SUBTYPING (*)

by E. Poll (1) (2), C. Hemerik (2) and H. M. M. Ten Eikelder (2)

Communicated by G. Longo

Abstract. – *In this paper we present constructions of cpo models for second order lambda calculi with recursive types and/or subtyping, that are compatible with conventional denotational semantics.*

For each of the systems we consider, the general structure of an environment model for that system is described first. For the systems with subtyping we prove coherence, i. e. *that the meaning of a term is independent of which particular type derivation we consider. The actual model constructions are then based on a standard fixed-point result for* ω*-categories. The combination and interaction of recursive types and subtyping does not pose any problems.*

Résumé. – *Dans cet article nous présentons une construction des modèles cpo pour les lambda calculs du second ordre à types récursifs et/ou sous-typage, qui sont compatibles avec la sémantique dénotationnelle conventionnelle.*

Pour chacun des systèmes que nous considérons, la structure générale d'un modèle d'environnement pour ce système est d'abord écrit. Pour les systèmes avec sous-typage nous prouvons leur cohérence, c'est-à-dire que nous montrons que le sens donné au terme ne dépend pas de la façon particulière de le dériver. En ce qui concerne les constructions des modèles, elles reposent sur un résultat classique de point fixe dans les ω*-catégories. La combinaison et l'interaction des types récursifs et du sous-typage ne pose pas de problème particulier.*

1. INTRODUCTION

The second order lambda calculus (or polymorphic lambda calculus) was discovered independently by Girard [Gir72] and Reynolds [Rey74]. It is an extension of the simple typed lambda calculus: not only terms but also types can be passed as parameters. This means that besides abstraction over term variables and application of terms to terms we also have abstraction over type variables and application of terms to types.

(*) Received May 1991, accepted June 1992.
(1) Supported by the Dutch organization for scientific research (NWO).
(2) Eindhoven University of Technology Department of Mathematics and Computing Science P.O. Box 513, Eindhoven, the Netherlands.

Both subtyping and recursive types are interesting extensions of second order lambda calculus from the point of view of programming languages. Recursive types can be used to make types such as lists and trees. Also fixed point operators, which cannot be typed in second order lambda calculus, can be typed using recursive types. Subtyping, in combination with labelled records, roughly corresponds with *inheritance* in object-oriented languages. This form of subtyping can be found in Cardelli and Wegner's language *Fun* [CW85], and more recently also in *Quest* [CL90].

Several models for second order lambda calculus are known, for example models based on partial equivalence relations [Gir72], the closure model [McC79], the finitary projection model [ABL86] and models based on qualitative domains [Gir86].

The models in this paper are more oriented towards programming language semantics, and are compatible with conventional denotational semantics. Types will be interpreted as cpos, which are commonly used as semantic domains in denotational semantics. Directed cpos or complete lattices could also be used. Recursion at term level can then be handled by the usual fixed point theory for cpos. Because types are interpreted as cpos we do not have empty types. Other type constructors, such as Σ (existential types), $\times$(Cartesian product), $+$(separated sum), $\otimes$(smashed product), $\oplus$(coalesced sum) or $(-)_\perp$ (lifting) can easily be added.

Providing a semantics for systems which have both subtyping and recursive types has long been regarded as problematic. Models that incorporate subtyping based on partial equivalence relations, such as Bruce and Longo's model for *Fun* [BL90] and Cardelli and Longo's model for (a part of) *Quest* [CL90], cannot easily be extended to model recursive types. Using the method described in [BCGS91] however, a semantics for subtyping and recursive types (but not for subtyping on recursive types) can be constructed using a semantics that models recursive types but does not model subtyping. For the model we construct the combination and interaction of recursive types and subtyping does not pose any problems. There will be no need to restrict the recursive types to those without negative occurences of the type variable.

Of the several versions of second order lambda calculus that can be found in literature, we here consider λ_2 [Bar9+], which contains the essential elements. In section 2 we briefly describe λ_2 and we give a general model definition for λ_2 based on the definition of a Bruce-Meyer-Mitchell environment model given in [BMM90]. Then we give a construction of a model that fits the general model definition.

In section 3 we consider λ_2 extended with recursive types, $\lambda_2\mu$, in section 4 λ_2 extended with subtyping, $\lambda_2\leqq$, and in section 5 we present a model for $\lambda_2\mu\leqq$, λ_2 with recursive types and subtyping. For each system we adapt the general model definition, and we then change the model construction accordingly.

For λ_2 and $\lambda_2\mu$ the model constructions are slight modifications of the construction given in [tEH89a]. Constructing a model is a question of solving the set of recursive domain equations given by the general model definition. Because types are interpreted as cpos, we can use the standard technique described in [SP82], and find a solution of the set of recursive domain equations by an inverse-limit construction in a product category.

Coercion functions are used to give the semantics of subtyping: if a type σ is a subtype of a type τ, we have a coercion function from the cpo for σ to the cpo for τ. The main problem in giving a model for systems with subtyping is that meanings are defined by induction on type derivations, and because of the subtyping many type derivations will be possible. As in [BCGS91], [BL90] and [CG90], we have to prove *coherence*, *i.e.* that all derivations for a term give the same meaning. For the systems with subtyping, $\lambda_2\leqq$ and $\lambda_2\mu\leqq$, we not only have to solve the recursive domain equations, but we also have to find coercion functions between the domains of types that are in the subtype relation. For the semantics to be coherent, the coercions have to satisfy certain conditions. Together, the domains and coercions form a *functor* from the subtype relation on types viewed as a category to *CPO*. Such a functor, satisfying both the recursive domain equations and the coherence conditions, is again found by an inverse-limit construction, only this time in a functor category.

2. λ_2

2.1. Syntax

We distinguish two sorts of expressions: types and terms.

Types

Let $\mathscr{V}_{\text{type}}$ be a set of type variables and $\mathscr{C}_{\text{type}}$ a set of type constants, or base types (e.g. bool, int or real). The set of types over $\mathscr{C}_{\text{type}}$ and $\mathscr{V}_{\text{type}}$ is given by:

$$\sigma = c \mid \alpha \mid \sigma_1 \rightarrow \sigma_2 \mid (\Pi\, \alpha : * . \sigma)$$

where $c \in \mathscr{C}_{\text{type}}$ and $\alpha \in \mathscr{V}_{\text{type}}$. We write "$\sigma : *$" for "$\sigma$ is a type".

Terms

Let $\mathscr{V}_{\text{term}}$ be a set of term variables and $\mathscr{C}_{\text{term}}$ be a set of term constants. All term constants have a specified type, which we will write as a superscript when necessary. We first define the set of *pseudo*-terms over $\mathscr{C}_{\text{term}}$ and $\mathscr{V}_{\text{term}}$, of which the set of terms will be a subset. The set of pseudo-terms over $\mathscr{C}_{\text{term}}$ and $\mathscr{V}_{\text{term}}$ is given by:

$$M = c \mid x \mid (\lambda x : \sigma . M) \mid M_1 M_2 \mid (\Lambda \alpha : * . M) \mid M \sigma$$

where $x \in \mathscr{V}_{\text{term}}$, $c \in \mathscr{C}_{\text{term}}$, $\alpha \in \mathscr{V}_{\text{type}}$ and σ a type.

So we have abstraction over term variables, $(\lambda x : \sigma . M)$, and we have abstraction over type variables, $(\Lambda \alpha : * . M)$, and the corresponding forms of application: of a term to a term, $M_1 M_2$, and of a term to a type, $M \sigma$.

Terms are those pseudo-terms for which a type can be derived in a context. A context is a syntactic type assignment of the form $x_0 : \sigma_0, \ldots, x_n : \sigma_n$, *i.e.* a partial function from $\mathscr{V}_{\text{term}}$ to the set of types. We write $\Gamma \vdash \mathrm{M} : \sigma$ if we can derive that in context Γ the term M has type σ, using the following rules:

$$\frac{c^{\sigma} \in \mathscr{C}_{\text{term}}}{\Gamma \vdash c^{\sigma} : \sigma} \qquad \frac{(x : \sigma) \in \Gamma}{\Gamma \vdash x : \sigma}$$

$$\frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash (\lambda x : \sigma . M) : \sigma \to \tau} (\to I) \qquad \frac{\Gamma \vdash M : \sigma \to \tau \quad \Gamma \vdash N : \sigma}{\Gamma \vdash MN : \tau} (\to E)$$

$$\frac{\Gamma \vdash M : \tau \quad \alpha \in \mathscr{V}_{\text{type}} \quad \alpha \text{ not free in } \Gamma}{\Gamma \vdash (\Lambda \alpha : * . M) : (\Pi \alpha : * . \tau)} (\Pi I) \qquad \frac{\Gamma \vdash M : (\Pi \alpha : * . \tau) \quad \sigma : *}{\Gamma \vdash M \sigma : \tau[\alpha := \sigma]} (\Pi E)$$

Term equality is the equality induced by the β and η rules (for both term and type abstraction and application).

The type of a term in a given context is unique, and equal terms have the same type (*see* [Bar9+]).

2.2. Semantics: general model definition

We now define the general structure of an environment model for λ_2. The definition is a particular instance of the one given in [BMM90]. It is simpler because we consider a simpler language. Another difference is that types are interpreted as cpos, whereas in [BMM90] types are interpreted as sets.

Before we can discuss the semantics of terms, we have to deal with unbound type variables in type expressions. Let Type be the set of closed type expressions. An environment η is a partial function from $\mathscr{V}_{\text{type}}$ to Type. The interpretation of a type σ in an environment η, written $[\![\sigma]\!]\eta$, is simply σ will all type variables α replaced by $\eta(\alpha)$. So for closed type expressions σ, *i.e.* $\sigma \in \text{Type}$, $[\![\sigma]\!]\eta = \sigma$.

For semantics of terms we associate a cpo Dom_σ with every $\sigma \in \text{Type}$. The meaning of $\Gamma \vdash M : \sigma$ in an environment η will be an element of the cpo $Dom_{[\![\sigma]\!]\eta}$. To define the semantics of abstraction and application these cpos have to satisfy certain requirements.

First we consider the cpos for function types.

Suppose $\Gamma \vdash M : \sigma \to \tau$. Then for all $\Gamma \vdash N : \sigma$ we have $\Gamma \vdash MN : \tau$, so we should be able to define the meaning of $MN (\in \text{Dom}_{[\![\tau]\!]\eta})$ in terms of the meanings of $M (\in \text{Dom}_{[\![\sigma \to \tau]\!]\eta})$ and $N (\in \text{Dom}_{[\![\sigma]\!]\eta})$. To get the meaning of MN, the meaning of M has to be considered as a mapping from $\text{Dom}_{[\![\sigma]\!]\eta}$ to $\text{Dom}_{[\![\tau]\!]\eta}$. So we require

$$\text{Dom}_{[\![\sigma \to \tau]\!]\eta} \cong [\text{Dom}_{[\![\sigma]\!]\eta} \to \text{Dom}_{[\![\tau]\!]\eta}] \qquad \text{(i)}$$

where the square brackets denote the subset of the function space containing the continuous functions. The isomorphism corresponding with (i), the bijection

$$\Phi_{[\![\sigma \to \tau]\!]\eta} \in \text{Dom}_{[\![\sigma \to \tau]\!]\eta} \to [\text{Dom}_{[\![\sigma]\!]\eta} \to \text{Dom}_{[\![\tau]\!]\eta}]$$

is the element-to-function mapping that we need to define the meaning of term abstraction and application. These mappings are similar to the element-to-function mappings that are used in models for the type-free λ-calculus.

For polymorphic types we need different mappings.

Suppose $\Gamma \vdash M : (\Pi\alpha : *.\tau)$. Then for all types σ we have $\Gamma \vdash M\sigma : \tau[\alpha := \sigma]$. By a simple substitution lemma $[\![\tau[\alpha := \sigma]]\!]\eta = [\![\tau]\!]\eta[\alpha := [\![\sigma]\!]\eta]$. So we should be able to define the meaning of

$$M\sigma\,(\in \text{Dom}_{[\![\tau[\alpha := \sigma]]\!]\eta} = \text{Dom}_{[\![\tau]\!]\eta[\alpha := [\![\sigma]\!]\eta]})$$

in terms of $[\![\sigma]\!]\eta\,(\in \text{Type})$ and the meaning of $M (\in \text{Dom}_{[\![\Pi\alpha : *.\tau]\!]\eta})$. This is achieved by requiring

$$\text{Dom}_{[\![\Pi\alpha : *.\tau]\!]\eta} \cong \prod_{a \in \text{Type}} \text{Dom}_{[\![\tau]\!]\eta[\alpha := a]} \qquad \text{(ii)}$$

Note that here we require more than is necessary: it would be sufficient if $\text{Dom}_{[\![\Pi\alpha : *.\tau]\!]\eta}$ were isomorphic to a subset of this product cpo containing only those polymorphic functions which are parametric, *i.e.* which behave in the same way for all types. For want of a definition of this subset we use the whole product cpo.

The isomorphism corresponding with (ii), the bijection

$$\Phi_{[\![\Pi\alpha:*.\tau]\!]\eta} \in \mathrm{Dom}_{[\![\Pi\alpha:*.\tau]\!]\eta} \to \prod_{a \in \mathrm{Type}} \mathrm{Dom}_{[\![\tau]\!]\eta[\alpha:=a]}$$

is used to define the meaning of type abstraction and application.

We now have domain equations for all function types and all polymorphic types. For the sake of a more uniform treatment, we also want a domain equation for the remaining types, the base types. For every base type σ a cpo domain$_\sigma$ has to be given. We could of course take Dom_σ equal to domain$_\sigma$, but instead we require

$$\mathrm{Dom}_\sigma \cong \mathrm{domain}_\sigma \qquad \text{(iii)}$$

For all $a \in \mathrm{Type}$, we define a function F_a that maps a family of cpos to a single cpo. If $\langle D_a \mid a \in \mathrm{Type} \rangle$ is a family of cpos, then

$$\begin{aligned} F_\sigma(\langle D_a \mid a \in \mathrm{Type}\rangle) &= \mathrm{domain}_\sigma && \text{for all } \sigma \in \mathscr{C}_{\mathrm{type}} \\ F_{\sigma\to\tau}(\langle D_a \mid a \in \mathrm{Type}\rangle) &= [D_\sigma \to D_\tau] && \text{for all } \sigma\to\tau \in \mathrm{Type} \\ F_{\Pi\alpha:*.\tau}(\langle D_a \mid a \in \mathrm{Type}\rangle) &= \prod_{a \in \mathrm{Type}} D_{\tau[\alpha:=a]} && \text{for all } (\Pi\alpha:*.\tau) \in \mathrm{Type} \end{aligned}$$

The system of coupled domain equations formed by (i), (ii) and (iii) can now be written as

$$\forall_{a \in \mathrm{Type}} : \mathrm{Dom}_a \cong F_a(\mathrm{Dom})$$

DEFINITION 1: (General model definition λ_2).

An environment model for λ_2 is a 3-tuple $\langle \mathrm{Dom}, \Phi_{\mathrm{term}}, \mathscr{I}_{\mathrm{term}} \rangle$, where

- $\mathrm{Dom} = \langle \mathrm{Dom}_a \mid a \in \mathrm{Type} \rangle$ is a family of cpos.
- $\Phi_{\mathrm{term}} = \langle \Phi_a \mid a \in \mathrm{Type} \rangle$ is a family of continuous bijections with $\Phi_a \in \mathrm{Dom}_a \to F_a(\mathrm{Dom})$, with the F_a defined as above.
- $\mathscr{I}_{\mathrm{term}} \in \mathscr{C}_{\mathrm{term}} \to \bigcup_{a \in \mathrm{Type}} \mathrm{Dom}_a$ gives the meanings of the term constants. Of course $\mathscr{I}_{\mathrm{term}}(c^\sigma) \in \mathrm{Dom}_\sigma$ for all $c^\sigma \in \mathscr{C}_{\mathrm{term}}$. □

If we can derive $\Gamma \vdash M : \sigma$, then $[\![\Gamma \vdash M : \sigma]\!]\eta$, the meaning of M with type σ in environment η, will be an element of $\mathrm{Dom}_{[\![\sigma]\!]\eta}$. Here an environment η is a funtion which gives the meaning of the free type variables in M and the term variables occurring in Γ, *i.e.* $\eta \in (\mathscr{V}_{\mathrm{type}} \cup \mathscr{V}_{\mathrm{term}}) \to (\mathrm{Type} \cup \bigcup_a \mathrm{Dom}_a)$.

We say that an environment η satisfies a context Γ if $\eta(\alpha)\in\text{Type}$ for all type variables α and $\eta(x)\in\text{Dom}_{[\![\sigma]\!]\eta}$ for all $x:\sigma$ in Γ.

For these environments we define the semantics of term expressions, by induction on their type derivation, as follows:

$$[\![\Gamma\vdash x:\sigma]\!]\eta=\eta(x)$$

$$[\![\Gamma\vdash c:\sigma]\!]\eta=\mathscr{I}_{\text{term}}(c)$$

$$[\![\Gamma\vdash MN:\tau]\!]\eta=(\Phi_s[\![\Gamma\vdash M:\sigma\to\tau]\!]\eta)[\![\Gamma\vdash N:\sigma]\!]\eta$$

$$[\![\Gamma\vdash(\lambda x:\sigma.M):\sigma\to\tau]\!]\eta=\Phi_s^{-1}(\lambda\xi\in\text{Dom}_{[\![\sigma]\!]\eta}.[\![\Gamma,\,x:\sigma\vdash M:\tau]\!]\eta[x:=\xi])$$

$$[\![\Gamma\vdash M\sigma:\tau]\!]\eta=(\Phi_t[\![\Gamma\vdash M:\Pi\alpha:\ast.\tau]\!]\eta)[\![\sigma]\!]\eta$$

$$[\![\Gamma\vdash(\Lambda\alpha:\ast.M):(\Pi\alpha:\ast.\tau)]\!]\eta=\Phi_t^{-1}(\lambda a\in\text{Type}.[\![\Gamma\vdash M:\tau]\!]\eta[\alpha:=a])$$

Here s is $[\![\sigma\to\tau]\!]\eta$ and t is $[\![\Pi\alpha:\ast.\tau]\!]\eta$.

For this general model definition we can prove type soundness, $[\![\Gamma\vdash M:\sigma]\!]\eta\in\text{Dom}_{[\![\sigma]\!]\eta}$, as well as soundness with respect to term equality, $\Gamma\vdash M=N:\sigma\Rightarrow[\![\Gamma\vdash M:\sigma]\!]\eta=[\![\Gamma\vdash N:\sigma]\!]\eta$ (*see* [BMM90]).

2.3. The construction of a cpo model

Because of the general model definition we have given, there only remains the task of finding a family of cpos $\text{Dom}=\langle\text{Dom}_a\mid a\in\text{Type}\rangle$, that solves the system of coupled domain equations:

$$\forall_{a\in\text{Type}}:\quad \text{Dom}_a\cong F_a(\text{Dom}) \tag{i}$$

with the associated bijections $\Phi_a\in\text{Dom}_a\to F_a(\text{Dom})$.

We use the standard technique, described in [SP82], to find a solution for the recursive domain equations. For this some category theory is needed.

An ω-chain is a diagram of the form $D_0\xrightarrow{f_0}D_1\xrightarrow{f_1}D_2\ldots$ An ω-category is a category with an initial object in which every ω-chain has a colimit. A functor is called ω-continuous if it preserves colimits of ω-chains. A fixed point of a functor $F:\mathscr{K}\to\mathscr{K}$ is a pair $(D,\,\varphi)$, where D is a $\mathscr{K}$-object and φ an isomorphism between D and $F(D)$.

The initial fixed point theorem ([SP82], [BH88]) states that for an ω-continuous functor on an ω-category an initial fixed point can be constructed, rather like for every continuous function on a cpo a least fixed point can be constructed. In fact, the fixed point theorem for cpos is a particular case of the initial fixed point theorem for ω-categories.

In denotational semantics this general result is usually applied to construct a solution of a single recursive domain equation $D \cong F(D)$. Because of the interdependence of the domain equations we have to solve them simultaneously. Therefore we shall construct a solution in a *product category*.

Product categories

Let I be an index set and C a category. The product category $\mathscr{K} = \prod_{a \in I} C$ is then defined as follows:

- objects of $\mathscr{K}$ are families $\langle D_a \mid a \in I \rangle$, where each D_a is a C-object;
- a $\mathscr{K}$-morphism from $\langle D_a \mid a \in I \rangle$ to $\langle E_a \mid a \in I \rangle$ is a family $\langle f_a \mid a \in I \rangle$, where each f_a is a C-morphism from D_a to E_a.

LEMMA 2: *If C is an ω-category, then so is $\prod_{a \in I} C$.*

Proof: Product categories are a special case of functor categories: the product category $\prod_{a \in I} C$ is a functor category C^J, where J is the discrete category with $\mathrm{Obj}(J) = I$. By [HS73] corollary 25.7, if in C is an ω-category then so is C^J, for any category J. □

For every $b \in I$ we have a projection functor P_b from $\mathscr{K}$ to C, which selects the b-component of a $\mathscr{K}$-object or morphism, *i.e.* $P_b(\langle X_a \mid a \in \mathrm{Type} \rangle) = X_b$.

LEMMA 3: *The projection functors are ω-continuous.*

Proof: Colimits in a product-category may be calculated pointwise ([HS73] theorem 25.6). This means that the projection functors preserve colimits. □

A functor F from $\mathscr{K}$ to $\mathscr{K}$ can be considered as a family of functors $\langle F_a \mid a \in I \rangle$, where every F_a is a functor from $\mathscr{K}$ to C.

LEMMA 4: *F is ω-continuous iff. every component F_a is ω-continuous.*

Proof: $F_a = P_a \circ F$, and by the previous lemma P_a is ω-continuous. So if F is ω-continuous, so are all the F_a. The reverse implication follows from the fact that colimits may be calculated pointwise ([HS73] theorem 25.6). □

Tupling of functors is denoted by $\langle\ ,\ \rangle$. For example,

$$\langle P_a, P_b \rangle : \mathscr{K} \to C \times C$$

is the functor which selects the a and b components of a $\mathscr{K}$-object or morphism.

The model construction

$\underline{CPO}$ is the category with cpos as objects and continuous functions as morphisms.

For the domain equations for function types we have the *function space functor*, *FS*, defined by

- $FS: \underline{CPO}^{OP} \times \underline{CPO} \to \underline{CPO}$;
- if D and E are cpos, then $FS(D, E) = [D \to E]$, the cpo of continuous functions from D to E, with the ordering pointwise;
- if $f \in [D' \to D]$ and $g \in [E \to E']$, then

$$FS(f, g) = (\lambda \xi \in [D \to E] . g \circ \xi \circ f) \in [[D \to E] \to [D' \to E']]$$

For the polymorphic types we have the *generalized product functor*, *GP*, defined by

- $GP: \prod_{a \in I} \underline{CPO} \to \underline{CPO}$.
- If $\langle D_a \mid a \in I \rangle$ is a family of cpos, then $GP(\langle D_a \mid a \in I \rangle) = \prod_{a \in I} D_a$, the cpo which is the product of all the cpos D_a, with the ordering coordinatewise.
- If $\langle f_a \mid a \in I \rangle$ is a family of functions, where $f_a \in [D_a \to E_a]$ for all $a \in I$, then

$$GP(\langle f_a \mid a \in I \rangle) = \lambda \langle d_a \mid a \in I \rangle \in GP(\langle D_a \mid a \in I \rangle) . \langle f_a(d_a) \mid a \in I \rangle$$

which is a continuous function from $GP(\langle D_a \mid a \in I \rangle)$ to $GP(\langle E_a \mid a \in I \rangle)$.

Because of the contravariance of FS in its first argument we cannot solve the recursive domain equations in the category $\prod_{a \in \text{Type}} \underline{CPO}$.

This problem is overcome using the standard technique. In [SP82] a theory of O-categories, a special class of categories, is developed. For an O-category C there is an associated category of embedding-projection pairs C_{PR}, and given a functor F on an O-category C, a corresponding functor F_{PR} on the category C_{PR} can be defined, which is covariant in all its arguments.

$\underline{CPO}$ is an O-category. The associated category of embedding-projection pairs is $\underline{CPO}_{PR}$, which is the category with cpos as objects and embedding-projection pairs as morphisms. An embedding-projection pair from cpo A to cpo B is a pair (φ, ψ) of continuous functions, $\varphi : A \to B$ and $\psi : B \to A$, such that $\psi \circ \varphi = \mathrm{id}_A$ and $\varphi \circ \psi \sqsubseteq \mathrm{id}_B$. $\underline{CPO}_{PR}$ is an ω-category (*see* [SP82], [BH88]).

The functors corresponding with FS and GP are

$$FS_{PR}: \underline{CPO}_{PR} \times \underline{CPO}_{PR} \to \underline{CPO}_{PR} \quad \text{and} \quad GP_{PR}: \prod_{a \in I} \underline{CPO}_{PR} \to \underline{CPO}_{PR}.$$

Note that FS_{PR} is covariant in both arguments. They are defined as follows

$$FS_{PR}(D, E) = FS(D, E)$$
$$FS_{PR}((\varphi, \psi), (\varphi', \psi')) = (FS(\psi, \varphi'), FS(\varphi, \psi'))$$

and

$$GP_{PR}(\langle D_a \mid a \in I \rangle) = GP(\langle D_a \mid a \in I \rangle)$$
$$GP_{PR}(\langle (\varphi_a, \psi_a) \mid a \in I \rangle) = (GP(\langle \varphi_a \mid a \in I \rangle), GP(\langle \psi_a \mid a \in I \rangle))$$

Note that the object parts are unchanged.

LEMMA 5: *FS_{PR} and GP_{PR} are ω-continuous.*

Proof: See [SP82] or [BH88] for FS_{PR}.

To prove ω-continuity for GP_{PR} we use the standard technique described in [SP82]. (In [SP82] instead of embedding-projection pairs just the embeddings are used. However, there is no essential difference, as every embedding uniquely determines the associated projection.) First we show that GP is *locally continuous*, which means that GP is continuous when viewed as a map from hom-sets in $\prod_{a \in I} \underline{CPO}$ to hom-sets in $\underline{CPO}$, *i.e.* for all ascending chains $f^0 \sqsubseteq f^1 \sqsubseteq f^2 \sqsubseteq \ldots$ in one of the hom-sets of $\prod_{a \in I} \underline{CPO}$

$$\bigsqcup_{n \in \mathbb{N}} GP(\langle f_a^n \mid a \in I \rangle) = GP(\langle \bigsqcup_{n \in \mathbb{N}} f_a^n \mid a \in I \rangle)$$

This follows immediately from the definition of GP.

In $\underline{CPO}$ every ω-chain has a colimit. Then by [HS73] theorem 25.7, the same is true in $\prod_{a \in I} \underline{CPO}$ and then by the dual version of [SP82] corollary to theorem 2 this category has locally determined colimits of embeddings. Then by [SP82] theorem 3 GP_{PR} is ω-continuous. □

For the base types we need *constant functors.* If A is a cpo then $C_A: \mathscr{K} \to \underline{CPO}_{PR}$ is the functor which maps every $\mathscr{K}$-object to the cpo A, and every $\mathscr{K}$-morphism to the identity morphism on A, which in the category $\underline{CPO}_{PR}$ is the embedding-projection pair $((\lambda\xi \in A.\xi), (\lambda\xi \in A.\xi))$.

We construct Dom in the product category $\mathscr{K} = \prod_{a \in \text{Type}} \underline{CPO}_{PR}$.

We define $F : \mathscr{K} \to \mathscr{K}$ by

$$F = \langle F_a \mid a \in \text{Type} \rangle$$

where the functors $F_a : \mathscr{K} \to \underline{CPO}_{PR}$ are defined as follows

$$F_\sigma = C_{\text{domain}_\sigma} \quad \text{for all } \sigma \in \mathscr{C}_{\text{type}}$$

$$F_{\sigma \to \tau} = FS_{PR} \circ \langle P_\sigma, P_\tau \rangle \quad \text{for all } \sigma \to \tau \in \text{Type}$$

$$F_{\Pi\alpha:*.\tau} = GP_{PR} \circ \langle P_{\tau[\alpha:=a]} \mid a \in \text{Type} \rangle \quad \text{for all } (\Pi\alpha : *.\tau) \in \text{Type}$$

Since FS_{PR}, GP_{PR}, C_A and P_a are all ω-continuous, so are all the F_a and hence so is F. Then by the initial fixed point theorem an initial fixed point can be constructed.

Let (Dom, m) be a fixed point of F. Then m is an isomorphism from Dom to F(Dom) in $\Pi\,\underline{CPO}_{PR}$. Because everything is defined pointwise, this means that all its components $m_a = (\Phi_a, \Psi_a)$ are isomorphisms from Dom_a to F_a(Dom) in CPO_{PR} (*i.e.* $\Psi_a = \Phi_a^{-1}$). Then Dom solves the recursive domain equations, and the embedding $\Phi_a : \text{Dom}_a \to F_a(\text{Dom})$ are the bijections we need.

So an initial fixed point of F gives a family of cpos Dom that satisfies the recursive domain equations with the associated bijections.

Recapitulating,

- $\underline{CPO}_{PR}$ is an ω-category;
- $\prod_{a \in \text{Type}} \underline{CPO}_{PR}$ is an ω-category;
- FS_{PR}, GP_{PR}, C_A and P_a are ω-continuous;
- for all $a \in \text{Type}$ the functor $F_a : \prod \underline{CPO}_{PR} \to \underline{CPO}_{PR}$ is ω-continuous;
- the functor $F = \langle F_a \mid a \in \text{Type} \rangle : \prod \underline{CPO}_{PR} \to \prod \underline{CPO}_{PR}$ is ω-continuous
- in $\prod_{a \in \text{Type}} \underline{CPO}_{PR}$ the equation $D \cong F(D)$ has an initial solution (Dom, m)

where $\text{Dom} = \langle \text{Dom}_a \mid a \in \text{Type} \rangle$ and $m = \langle m_a \mid a \in \text{Type} \rangle$

- $m_a = (\Phi_a, \Psi_a)$ is an isomorphism between Dom_a and F_a(Dom) for all $a \in \text{Type}$.

3. RECURSIVE TYPES

We now extend λ_2 with recursive types, resulting in the system $\lambda_2\mu$. The set of types over $\mathscr{C}_{\text{type}}$ and $\mathscr{V}_{\text{type}}$ is now given by:

$$\sigma = c \mid \alpha \mid \sigma_1 \to \sigma_2 \mid (\Pi\alpha : \ast . \sigma) \mid (\mu\alpha : \ast . \sigma)$$

where $c \in \mathscr{C}_{\text{type}}$ and $\alpha \in \mathscr{V}_{\text{type}}$.

A recursive type $(\mu\alpha : \ast . \sigma)$ is considered as a solution of

$$(\mu\alpha : \ast . \sigma) \approx \sigma[\alpha := (\mu\alpha : \ast . \sigma)]$$

We interpret recursive types as infinite types, so all recursive types that have the same infinite unfolding are identified. For example, suppose we have a term M of type $(\mu\alpha : \ast . \alpha \to \text{int})$. Because

$$(\mu\alpha : \ast . \alpha \to \text{int}) \approx (\mu\alpha : \ast . \alpha \to \text{int}) \to \text{int},$$

we can apply M to itself, and the result should be of type int.

There are other ways to treat recursive types:

• A recursive type $(\mu\alpha : \ast . \sigma)$ and its unfolding $\sigma[\alpha := (\mu\alpha : \ast . \sigma)]$ are not identified. We introduce explicit coercion operators $\text{fold}_{(\mu\alpha:\ast.\sigma)}$ and $\mathit{unfold}_{(\mu\alpha:\ast.\sigma)}$ and we add the rules

$$\frac{\Gamma \vdash M : (\mu\alpha : \ast . \sigma)}{\Gamma \vdash \text{unfold}_{(\mu\alpha : \ast . \sigma)} M : \sigma[\alpha := (\mu\alpha : \ast . \sigma)]} \qquad \frac{\Gamma \vdash M : \sigma[\alpha := (\mu\alpha : \ast . \sigma)]}{\Gamma \vdash \text{fold}_{(\mu\alpha : \ast . \sigma)} M : (\mu\alpha : \ast . \sigma)}$$

For the model we then require

$$\text{Dom}_{[\![(\mu\alpha : \ast . \sigma)]\!]\eta} \cong \text{Dom}_{[\![\sigma[\alpha := (\mu\alpha : \ast . \sigma)]]\!]\eta}$$

The associated isomorphism gives the meaning of the fold and unfold operators.

• We define equality as the congruence relation induced by $(\mu\alpha : \ast . \sigma) = \sigma[\alpha := (\mu\alpha : \ast . \sigma)]$. This means that a recursive type and its unfoldings are identified, but for example the types $(\mu\alpha : \ast . \alpha \to \text{int})$ and $(\mu\alpha : \ast . (\alpha \to \text{int}) \to \text{int})$ are not be identified, because by unfolding them we can never get the same type: unfolding the first type gives $((\ldots(\mu\alpha : \ast . \alpha \to \text{int})\ldots \to \text{int}) \to \text{int}$ and unfolding the second type gives $((\ldots((\mu\alpha : \ast . (\alpha \to \text{int}) \to \text{int})\ldots \to \text{int}) \to \text{int}$. Considered as infinite types however, these types are equal.

For these two possibilities the general model definition and the model construction for λ_2 can also be adapted (*see* [Pol91]).

3.1. Syntax

When we consider recursive types as finite types, we get a congruence relation on types. To define this relation, we define a tree $\mathcal{T}(\sigma)$ for every type σ, as is done in [CC91] for the simple typed lambda calculus with recursive types. These trees will be *regular* trees, *i.e.* trees with a finite set of subtrees. The leaves are base types or type variables, and the nodes correspond to type constructors.

DEFINITION 6: *Tree* is the set of all trees with base types, type variables and $\perp$ as leaves, and $\rightarrow$ and $\Pi_\alpha, \alpha \in \mathcal{V}_{type}$, as nodes. $\rightarrow$-nodes have two subtrees, Π_α-nodes have one subtree. □

Note that we have bound type variables in the trees: every Π-node introduces a bound type variable. α-equal trees are identified. We want $\mathcal{T}(\mu\alpha : \ast . \sigma)$ to be a solution of $x = \mathcal{T}(\sigma)[\alpha := x]$. By the following property this equation has a unique solution for all $\mathcal{T}(\sigma) \neq .\alpha$.

PROPERTY 7: (Cou83], theorem 4.3.1): If $t \neq .\alpha$ and t is regular, then there is a unique tree x such that $x = t[\alpha := x]$, and this tree x is regular. □

$\mathcal{T}(\mu\alpha : \ast . \alpha)$ will be $\perp$. To be able to prove properties of trees by induction we define a partial order $\sqsubseteq$ on *Tree*.

DEFINITION 8: $\sqsubseteq$ is a partial order on Tree, defined by

$$\perp \sqsubseteq s \quad \text{for all } s \in \text{Tree} \qquad s \sqsubseteq s \quad \text{for all } s \in \text{Tree}$$

$$\begin{tikzcd}[column sep=small] & \rightarrow \arrow[dl, dash] \arrow[dr, dash] & & \sqsubseteq & & \rightarrow \arrow[dl, dash] \arrow[dr, dash] & \\ s & & t & & s' & & t' \end{tikzcd} \quad \text{if } s \sqsubseteq s' \text{ and } t \sqsubseteq t' \qquad \begin{tikzcd}[column sep=small] \Pi_\alpha \arrow[d] & \sqsubseteq & \Pi_\alpha \arrow[d] \\ s & & t \end{tikzcd} \quad \text{if } s \sqsubseteq t \quad \square$$

So $a \sqsubseteq b$ if we can get a by cutting of some subtrees of b and replacing them by $\perp$. (Tree, $\sqsubseteq$) is a cpo.

DEFINITION 9: The function $\mathscr{T}$ from types to regular trees is defined by

$$\mathscr{T}(\sigma) = .\sigma \quad \text{if } \sigma \in \mathscr{C}_{\text{type}} \cup \mathscr{V}_{\text{type}}$$

$$\mathscr{T}(\sigma \to \tau) = \begin{tikzcd} & \to \arrow[dl, dash] \arrow[dr, dash] & \\ \mathscr{T}(\sigma) & & \mathscr{T}(\tau) \end{tikzcd}$$

$$\mathscr{T}(\Pi\alpha : \ast . \sigma) = \begin{tikzcd} \Pi_\alpha \arrow[d] \\ \mathscr{T}(\sigma) \end{tikzcd}$$

$$\mathscr{T}(\mu\alpha : \ast . \sigma) = \underline{fix}(\lambda t \in \text{Tree} . \mathscr{T}(\sigma)[\alpha := t]) \quad \square$$

$(\lambda t \in \text{Tree} . \mathscr{T}(\sigma)[\alpha := t])$ is a continuous function. Its least fixed point is the smallest solution of $x = \mathscr{T}(\sigma)[\alpha := x]$. This solution is regular; if $\mathscr{T}(\sigma) = .\alpha$ it is $\perp$, else property 7 applies.

DEFINITION 10: Type equality, written $\approx$, is defined by

$$\sigma \approx \tau \quad \Leftrightarrow \quad \mathscr{T}(\sigma) = \mathscr{T}(\tau) \quad \square$$

We add the type conversion rule $(\approx)$:

$$\frac{\Gamma \vdash M : \sigma \quad \sigma \approx \tau}{\Gamma \vdash M : \tau}(\approx)$$

3.2. Semantics: general model definition

Types are interpreted as trees. The leaves are base types or type variables, and the nodes correspond to type constructors. The meaning of a type σ in environment η is the tree $\mathscr{T}(\sigma)$ with all free type variables α replaced by $\eta(\alpha)$, *i.e.*

$$[\![\sigma]\!]\eta = \mathscr{T}(\sigma)[\alpha_0 := \eta(\alpha_0), \ldots, \alpha_n := \eta(\alpha_n)]$$

where $\{\alpha_0, \ldots, \alpha_n\}$ is the set of free type variables of σ. It is an element of the following set Type:

DEFINITION 11: $\text{Type} = \{ t \in \text{Tree} \mid t \text{ is regular } \wedge FV(t) = \emptyset \}$. Here $FV(t)$ denotes the set of free type variables occurring in a tree t. $\square$

If $\sigma \approx \tau$, then $\mathscr{T}(\sigma) = \mathscr{T}(\tau)$, and hence $[\![\sigma]\!]\eta = [\![\tau]\!]\eta$ for all environments η. We can take the same recursive domain equations we had for λ_2:

$$\forall a \in \text{Type}: \quad \text{Dom}_a \cong F_a(\text{Dom})$$

where

$$F_{.\sigma}<D_a \mid a\in\text{Type}\rangle)=\text{domain}_\sigma \quad \text{for } \sigma\in\mathscr{C}_{\text{type}}$$

$$F_{\substack{\to \\ \diagup\ \diagdown \\ \sigma\qquad\tau}}\ (\langle D_a \mid a\in\text{Type}\rangle)=[D_\sigma\to D_\tau]$$

$$F_{\substack{\Pi_\alpha \\ \downarrow \\ \tau}}(\langle D_a \mid a\in\text{Type}\rangle)=\prod_{a\in\text{Type}} D_{\tau[\alpha:=a]}$$

$$F_\perp(\langle D_a \mid a\in\text{Type}\rangle)=\text{domain}_\perp=\{\perp\}$$

So $\text{Dom}_{[\![(\mu\alpha:*.\alpha)]\!]\eta}=\text{Dom}_\perp$ is the one-point cpo.

DEFINITION 12 (General model definition $\lambda_2\mu$):

An environment model for $\lambda_2\mu$ is defined as for λ_2 (definition 1), except with Type, $[\![\ ]\!]$ for type expressions and $F=\langle F_a \mid a\in\text{Type}\rangle$ defined as above. □

Remember that the meaning of a term is defined by induction on its type derivation and the that type inference rule ($\approx$) has been added. We define $[\![\Gamma\vdash M:\tau]\!]\eta=[\![\Gamma\vdash M:\sigma]\!]\eta$ if $\Gamma\vdash M:\tau$ follows from $\Gamma\vdash M:\sigma$ by ($\approx$).

Because of the rule ($\approx$), there may now be more than one way to derive $\Gamma\vdash M:\sigma$. We have to prove *coherence*, *i.e.* that all derivations for $\Gamma\vdash M:\sigma$ give the same meaning $[\![\Gamma\vdash M:\sigma]\!]\eta$. The same problem will occurs in the next section, when we introduce subtyping, but there it will be more complicated. Here coherence can easily be proved. We can show that terms have a unique type modulo $\approx$ and that terms have a unique meaning.

LEMMA 13: *If* $\Gamma\vdash M:\sigma$ *and* $\Gamma\vdash M:\tau$ *then* $\sigma\approx\tau$ (*and so* $[\![\sigma]\!]\eta=[\![\tau]\!]\eta$ *for all* η).

Proof: Induction on the structure of M. □

LEMMA 14: *Every term has a unique meaning*, i.e.

$$[\![\Gamma\vdash M:\sigma]\!]\eta=[\![\Gamma\vdash M:\tau]\!]\eta$$

for all possible derivations $\Gamma\vdash M:\sigma$ *and* $\Gamma\vdash M:\tau$.

Proof: Induction on the structure of M. We treat only one case, viz. $M=N_1 N_2$; the others are similar. Suppose $\Gamma\vdash N_1 N_2:\tau$. By the IH N_1 and N_2 have a unique meaning, say X_1 and X_2, respectively. Any derivation of $\Gamma\vdash N_1 N_2:\tau$ must end with ($\to E$), followed by zero or more uses of the rule ($\approx$). Because ($\approx$) does not affect the meaning of terms, this yields

$[\![\Gamma \vdash N_1 N_2 : \tau]\!] \eta = (\Phi_s X_1) X_2$, where s is the (by the previous lemma unique) meaning of the type of N_1. □

3.3. The construction of a cpo model

To complete the model, we have to construct a family of cpos Dom that solves the system of coupledd domain equations:

$$\forall a \in \text{Type}: \quad \text{Dom}_a \cong F_a(\text{Dom})$$

We define the functor $F: \mathscr{K} \to \mathscr{K}$ by $F = \langle F_a \mid a \in \text{Type} \rangle$, where the functors $F_a: \mathscr{K} \to \underline{CPO_{PR}}$ are defined by

$$F_{.\sigma} = C_{\text{domain}_\sigma} \qquad \text{for .} \ \sigma \in \mathscr{C}_{\text{type}}$$

$$F_{\overset{\rightarrow}{\sigma \ \ \tau}} = FS_{PR^0} \langle P_\sigma, P_\tau \rangle$$

$$F_{\overset{\Pi_\alpha}{\overset{\downarrow}{\tau}}} = GP_{PR^0} \langle P_{\tau[\alpha := a]} \mid a \in \text{Type} \rangle$$

$$F_\perp = C_{\text{domain}_\perp}$$

The initial fixed point of F gives the cpos Dom_a satisfying the recursive domain equations, and the associated isomorphisms $\Phi_a \in \text{Dom}_a \to F_a(\text{Dom})$.

4. SUBTYPING

We now consider the extension of system λ_2 with subtyping. This system is called $\lambda_2 \leqq$. For case of presentation, we consider a very simple form of subtyping, which is based on a subtype relation between base types. In section 6 we will indicate how labelled records and bounded quantification, which are more interesting forms of subtyping, can be dealt with is a similar way.

4.1. Syntax

We have a subtype relation $\leqq$ on types. If $\sigma \leqq \tau$, we say that σ is a subtype of τ. The subtype relation will be a pre-order (*i.e.* reflexive and transitive).

We add the following type inference rule: the *subsumption rule*

$$\frac{\Gamma \vdash M:\sigma \quad \sigma \leqq \tau}{\Gamma \vdash M:\tau}(SUB)$$

All subtyping is based on a reflexive and transitive subtype relation $\leqq^B$ on the base types. For example, if int and real are base types, we could have $\text{int} \leqq^B \text{real}$.

We have the following rules for deducing $\sigma \leqq \tau$

$$\frac{\sigma \leqq^B \tau}{\sigma \leqq \tau}(START) \qquad \frac{\sigma:*}{\sigma \leqq \sigma}(REFL) \qquad \frac{\rho \leqq \sigma \quad \sigma \leqq \tau}{\rho \leqq \tau}(TRANS)$$

$$\frac{\sigma' \leqq \sigma \quad \tau \leqq \tau'}{\sigma \to \tau \leqq \sigma' \to \tau'}(\leqq \to) \qquad \frac{\sigma \leqq \tau}{(\Pi\alpha:*.\sigma) \leqq (\Pi\alpha:*.\tau)}(\leqq \Pi)$$

Note the contravariance of $\to$ with respect to the subtype relation. That $\leqq$ is indeed a pre-order is of course guaranteed by the rule (*REFL*) and (*TRANS*). In fact, we do not need (*TRANS*).

LEMMA 15: *The rule* (*TRANS*) *is derivable.*

Proof: A straightforward induction on derivations proves that if $\rho \leqq \sigma$ and $\sigma \leqq \tau$ can be derived without using (*TRANS*), so can $\rho \leqq \tau$. Here it is essential that $\leqq^B$ is already transitive. □

4.2. Semantics: general model definition

As for λ_2, Type is simply the set of closed type expressions.

Because the semantics of terms is defined by induction on type derivations, we have to define the semantics of the new type inference rule, the subsumption rule. Suppose $\Gamma \vdash M:\tau$ is derived from $\Gamma \vdash M:\sigma$ and $\sigma \leqq \tau$. Since $[\![\Gamma \vdash M:\sigma]\!]\eta \in \text{Dom}_{[\![\sigma]\!]\eta}$ and we want $[\![\Gamma \vdash M:\tau]\!]\eta \in \text{Dom}_{[\![\tau]\!]\eta}$, we need a *coercion function* from $\text{Dom}_{[\![\sigma]\!]\eta}$ to $\text{Dom}_{[\![\tau]\!]\eta}$. We call this function $\text{Coe}_{[\![\sigma]\!]\eta\,[\![\tau]\!]\eta}$.

We can now give the meaning of $M:\tau$ in terms of the meaning of $M:\sigma$

$$[\![\Gamma \vdash M:\tau]\!]\eta = \text{Coe}_{[\![\sigma]\!]\eta\,[\![\tau]\!]\eta}[\![\Gamma \vdash M:\sigma]\!]\eta$$

For all types σ and τ such that $\sigma \leqq \tau$, we need a coercion function from $\text{Dom}_{[\![\sigma]\!]\eta}$ to $\text{Dom}_{[\![\tau]\!]\eta}$. We require that the coercion functions are continuous.

Because of the rule (*SUB*) there may be more than one way to derive $\Gamma \vdash M:\sigma$. So we have to prove coherence, as we did for $\lambda_2\mu$. However, the problem is now more complicated, because terms no longer have a unique meaning. Not only will there be more than one type derivation for $\Gamma \vdash M:\sigma$, but in different derivations a subexpression of M may have different types and hence different meanings. To prove that all derivations for $\Gamma \vdash M:\sigma$ give the same meaning $[\![\Gamma \vdash M:\sigma]\!]\eta$, some additional requirements for the coercion functions are needed.

The following two requirements for the coercion functions are obvious:

$$\mathscr{P}_0 : \mathrm{Coe}_{[\![\sigma]\!]\eta\,[\![\sigma]\!]\eta} = \lambda\xi \in \mathrm{Dom}_{[\![\sigma]\!]\eta}\,.\,\xi \quad \text{for all } \sigma : *$$

$$\mathscr{P}_1 : \mathrm{Coe}_{[\![\rho]\!]\eta\,[\![\tau]\!]\eta} = \mathrm{Coe}_{[\![\sigma]\!]\eta\,[\![\tau]\!]\eta} \circ \mathrm{Coe}_{[\![\rho]\!]\eta\,[\![\sigma]\!]\eta} \quad \text{for all } \rho \leqq \sigma \leqq \tau$$

Clearly, if $\mathscr{P}_0$ or $\mathscr{P}_1$ does not hold, then the semantics is not coherent. $\mathscr{P}_0$ and $\mathscr{P}_1$ are not sufficient to have coherence. We will also require properties of the coercions between function types and polymorphic types.

First we consider function types. Suppose $\sigma \to \tau \leqq \sigma' \to \tau'$. For the sake of simplicity we assume that the types are closed, so that we can omit $[\![\ldots]\!]\eta$. Let $\Gamma \vdash M:\sigma \to \tau$ and $\Gamma \vdash N:\sigma'$. Then $\Gamma \vdash MN:\tau'$ can be derived in at least two ways:

$$\dfrac{\dfrac{M:\sigma\to\tau \quad \sigma\to\tau\leqq\sigma'\to\tau'}{M:\sigma'\to\tau'} \qquad N:\sigma'}{MN:\tau'} \qquad\qquad \dfrac{\dfrac{M:\sigma\to\tau \qquad \dfrac{N:\sigma' \quad \sigma'\leqq\sigma}{N:\sigma}}{MN:\tau} \qquad \tau\leqq\tau'}{MN:\tau'}$$

These two derivations give as $[\![\Gamma \vdash MN:\tau']\!]\eta$

$$(\Phi_{\sigma'\to\tau'}(\mathrm{Coe}_{\sigma\to\tau\,\sigma'\to\tau'}[\![\Gamma \vdash M:\sigma\to\tau]\!]\eta))\,[\![\Gamma \vdash N:\sigma']\!]\eta \qquad \text{(i)}$$

$$\mathrm{Coe}_{\tau\tau'}((\Phi_{\sigma\to\tau}[\![\Gamma \vdash M:\sigma\to\tau]\!]\eta)\,(\mathrm{Coe}_{\sigma'\sigma}[\![\Gamma \vdash N:\sigma']\!]\eta)) \qquad \text{(ii)}$$

In order for these to be equal, some equation between $\mathrm{Coe}_{\sigma\to\tau\,\sigma'\to\tau'}$ and $\mathrm{Coe}_{\sigma'\sigma}$ and $\mathrm{Coe}_{\tau\tau'}$ has to hold. There is really only one way to express a relation between $\mathrm{Coe}_{\sigma\to\tau\,\sigma'\to\tau'}$ and $\mathrm{Coe}_{\sigma'\sigma}$ and $\mathrm{Coe}_{\tau\tau'}$:

$$\begin{tikzcd}
\mathrm{Dom}_{\sigma\to\tau} \arrow[d, "\mathrm{Coe}_{\sigma\to\tau\,\sigma'\to\tau'}"] & \cong FS(\mathrm{Dom}_\sigma, \mathrm{Dom}_\tau) \arrow[d, "FS(\mathrm{Coe}_{\sigma'\sigma}, \mathrm{Coe}_{\tau\tau'})"] & = [\mathrm{Dom}_\sigma \to \mathrm{Dom}_\tau] \\
\mathrm{Dom}_{\sigma'\to\tau'} & \cong FS(\mathrm{Dom}_{\sigma'}, \mathrm{Dom}_{\tau'}) & = [\mathrm{Dom}_{\sigma'} \to \mathrm{Dom}_{\tau'}]
\end{tikzcd}$$

$\mathscr{P}_2$: for all $\sigma \to \tau \leqq \sigma' \to \tau'$

$$\mathrm{Coe}_{[\![\sigma\to\tau]\!]\eta\,[\![\sigma'\to\tau']\!]\eta} = \Phi^{-1}_{[\![\sigma'\to\tau']\!]\eta} \circ FS(\mathrm{Coe}_{[\![\sigma']\!]\eta\,[\![\sigma]\!]\eta}, \mathrm{Coe}_{[\![\tau]\!]\eta\,[\![\tau']\!]\eta}) \circ \Phi_{[\![\sigma\to\tau]\!]\eta}$$

If $\mathscr{P}_2$ holds, then (i) and (ii) are equal.

Now we consider polymorphic types. Let $(\Pi\alpha:*.\sigma)$, $(\Pi\alpha:*.\tau)$ and ρ be closed types, $(\Pi\alpha:*.\sigma)\leqq(\Pi\alpha:*.\tau)$, and suppose $\Gamma\vdash M:(\Pi\alpha:*.\sigma)$. Then $\Gamma\vdash M\rho:\tau[\alpha:=\rho]$ can be derived in at least two ways:

$$\dfrac{\dfrac{M:(\Pi\alpha:*.\sigma)\quad(\Pi\alpha:*.\sigma)\leqq(\Pi\alpha:*.\tau)}{M:(\Pi\alpha:*.\tau)}\qquad\rho:*}{M\rho:\tau[\alpha:=\rho]}$$

$$\dfrac{\dfrac{M:(\Pi\alpha:*.\sigma)\quad\rho:*}{M\rho:\sigma[\alpha:=\rho]}\qquad\sigma[\alpha:=\rho]\leqq\tau[\alpha:=\rho]}{M\rho:\tau[\alpha:=\rho]}$$

These two derivations give for $[\![\Gamma\vdash M\rho:\tau[\alpha:=\rho]]\!]\eta$

$$(\Phi_{\Pi\alpha:*.\tau}(\mathrm{Coe}_{(\Pi\alpha:*.\sigma)(\Pi\alpha:*.\tau)}[\![\Gamma\vdash M:\Pi\alpha:*.\sigma]\!]\eta))\sigma \qquad \text{(iii)}$$

$$\mathrm{Coe}_{\sigma[\alpha:=\rho]\tau[\alpha:=\rho]}((\Phi_{\Pi\alpha:*.\sigma}[\![\Gamma\vdash M:\Pi\alpha:*.\sigma]\!]\eta)\rho) \qquad \text{(iv)}$$

Again, we want these to be equal. There is only one way we can express a relation between $\mathrm{Coe}_{(\Pi\alpha:*.\sigma)(\Pi\alpha:*.\tau)}$ and $\mathrm{Coe}_{\sigma[\alpha:=\rho]\tau[\alpha:=\rho]}$:

$$\begin{tikzcd}
\mathrm{Dom}_{\Pi\alpha:*.\sigma} \arrow[d, "\mathrm{Coe}_{(\Pi\alpha:*.\sigma)(\Pi\alpha:*.\tau)}"] \arrow[r, phantom, "\cong"] & GP(\langle \mathrm{Dom}_{\sigma[\alpha:=a]}\mid a\in\mathrm{Type}\rangle)=\prod_{a\in\mathrm{Type}}\mathrm{Dom}_{\sigma[\alpha:=a]} \arrow[d, "GP(\langle\mathrm{Coe}_{\sigma[\alpha:=a]\tau[\alpha:=a]}\mid a\in\mathrm{Type}\rangle"] \\
\mathrm{Dom}_{\Pi\alpha:*.\tau} \arrow[r, phantom, "\cong"] & GP(\langle \mathrm{Dom}_{\tau[\alpha:=a]}\mid a\in\mathrm{Type}\rangle)=\prod_{a\in\mathrm{Type}}\mathrm{Dom}_{\tau[\alpha:=a]}
\end{tikzcd}$$

$\mathscr{P}_3$: for all $(\Pi\alpha:*.\sigma)\leqq(\Pi\alpha:*.\tau)$

$$\mathrm{Coe}_{[\![\Pi\alpha:*.\sigma]\!]\eta\,[\![\Pi\alpha:*.\tau]\!]\eta}=\Phi^{-1}_{[\![\Pi\alpha:*.\tau]\!]\eta}\circ GP(\langle\mathrm{Coe}_{[\![\sigma]\!]\eta[\alpha:=a]\,[\![\tau]\!]\eta[\alpha:=a]}\mid a\in\mathrm{Type}\rangle)\circ\Phi_{[\![\Pi\alpha:*.\sigma]\!]\eta}$$

If $\mathscr{P}_3$ holds, then (iii) and (iv) are indeed equal.

The semantics is coherent, if and only if the coherence conditions $\mathscr{P}_0$, $\mathscr{P}_1$, $\mathscr{P}_2$ and $\mathscr{P}_3$ hold. The proof can be found in appendix A. For the proof we use the fact that we have *minimal typing* in $\lambda_2\leqq$.

DEFINITION 16 (General model definition $\lambda_2\leqq$):

An environment model for $\lambda_2\leqq$ is a 4-tuple $\langle\mathrm{Dom},\Phi_{\mathrm{term}},\mathscr{I}_{\mathrm{term}},\mathrm{Coe}\rangle$, where Coe is a family of coercion functions, $\mathrm{Coe}=\langle\mathrm{Coe}_{ab}\in[\mathrm{Dom}_a\to\mathrm{Dom}_b]\mid$ for all $a, b\in\mathrm{Type}, a\leqq b\rangle$, satisfying $\mathscr{P}_0$, $\mathscr{P}_1$, $\mathscr{P}_2$ and $\mathscr{P}_3$, and the rest as in definition 1. □

4.3 The construction of a cpo model

Before we can begin to construct a cpo-model for $\lambda_2 \leqq$, some coercions have to be given. We need coercion functions $\text{coerce}_{\sigma\tau}$ from domain_σ to domain_τ, for all base types σ and τ such that $\sigma \leqq^B \tau$. We require that these coercion functions are continuous, and that $\mathscr{P}_0$ and $\mathscr{P}_1$ hold, *i.e.*

$$\text{coerce}_{\sigma\sigma} = \lambda\xi \in \text{domain}_\sigma . \xi$$

$$\text{coerce}_{\rho\tau} = \text{coerce}_{\sigma\tau} \circ \text{coerce}_{\rho\rho} \quad \text{if} \quad \rho \leqq^B \sigma \leqq^B \tau$$

For $\sigma \leqq^B \tau$, $\text{Coe}_{\sigma\tau} \in [\text{Dom}_\sigma \to \text{Dom}_\tau]$ is of course defined by

$$\text{Coe}_{\sigma\tau} = \Phi_\tau^{-1} \circ \text{coerce}_{\sigma\tau} \circ \Phi_\sigma$$

So we are looking for a family of cpos $< \text{Dom}_a \mid a \in \text{Type} \rangle$, solving the coupled domain equations

$$\text{Dom}_\sigma \cong \text{domain}_\sigma$$

$$\text{Dom}_{\sigma \to \tau} \cong FS(\text{Dom}_\sigma, \text{Dom}_\tau)$$

$$\text{Dom}_{\Pi\alpha : *.\tau} \cong GP(\langle \text{Dom}_{\tau[\alpha := a]} \mid a \in \text{Type} \rangle)$$

and a family of coercion functions $\langle \text{Coe}_{ab} \mid a \leqq b \rangle$ satisfying $\mathscr{P}_0$, $\mathscr{P}_1$ and

$$\text{Coe}_{\sigma\tau} = \Phi_\tau^{-1} \circ \text{coerce}_{\sigma\tau} \circ \Phi_\sigma \quad \text{for all} \quad \sigma \leqq^B \tau$$

$$\text{Coe}_{\sigma \to \tau\ \sigma' \to \tau'} = \Phi_{\sigma' \to \tau'}^{-1} \circ FS(\text{Coe}_{\sigma'\sigma}, \text{Coe}_{\tau\tau'}) \circ \Phi_{\sigma \to \tau} \quad \text{for all} \quad \sigma \to \tau \leqq \sigma' \to \tau'$$

$$\text{Coe}_{(\Pi\alpha : *.\sigma)\,(\Pi\alpha : *.\tau)} = \Phi_{\Pi\alpha : *.\tau}^{-1} \circ GP(\langle \text{Coe}_{\sigma[\alpha := a]\,\tau[\alpha := a]} \mid a \in \text{Type} \rangle) \circ \Phi_{\Pi\alpha : *.\sigma}$$
$$\text{for all } (\Pi\alpha : \star . \sigma) \leqq (\Pi\alpha : \star . \tau)$$

We define $\underline{\text{Type}}$ as the subtype relation on Type viewed as a category.

DEFINITION 17: The objects of the category $\underline{\text{Type}}$ are the elements of Type, and there is a unique morphism, called $a \leqq b$, from a to b iff $a \leqq b$. Because $\leqq$ is reflexive, there is an identity $a \leqq a$ for all objects a. Because $\leqq$ is transitive, composition is always defined: $b \leqq c \circ a \leqq b$ is $a \leqq c$. □

Together, Dom and Coe can be seen as a *functor* from $\underline{\text{Type}}$ to $\underline{CPO}$. Dom is the object part, mapping every $\underline{\text{Type}}$-object, *i.e.* every element of Type, to a $\underline{CPO}$-object, a cpo. Coe is the morphism part, mapping every $\underline{\text{Type}}$-morphism $a \leqq b$ to a continuous function from Dom_a to Dom_b. For this to be a functor, identities and composition must be preserved. This is guaranteed by $\mathscr{P}_0$ and $\mathscr{P}_1$.

We construct Dom & Coe, the functor formed by Dom and Coe together, as an initial fixed point in a functor category. Because of the contravariance of FS in its first argument, we cannot construct Dom in the standard functor category $[\underline{\text{Type}}, \underline{CPO}]$ (usually written $\underline{CPO}^{\text{Type}}$). Instead, we work in the associated category of embedding-projection pairs. Morphisms of $[\text{Type}, \underline{CPO}]$ are natural transformations, families of $\underline{CPO}$-morphisms. So, pointwise, they have the same properties as CPO-morphisms, in particular those properties that enable the use of embedding-projection pairs.

$\underline{CPO}_\perp$ is the category with cpos as objects and *strict* continuous functions as morphisms. It is a subcategory of $\underline{CPO}$.

DEFINITION 18: $[\underline{\text{Type}}, \underline{CPO}_\perp]_{PR}$ is the category with as objects functors from $\underline{\text{Type}}$ to $\underline{CPO}_\perp$, and as morphisms embedding-projection pairs of natural transformations:
if F and G are functors from $\underline{\text{Type}}$ to $\underline{CPO}_\perp$, then (φ, ψ) is a morphism from F to G if

$$\varphi: \quad F \dot{\rightarrow} G \quad (i.e.\ \varphi \text{ is a natural transformation from } F \text{ to } G)$$

$$\psi: \quad G \dot{\rightarrow} F$$

and for all $a \in \text{Type}: \psi_a \circ \varphi_a = \text{id}_{Fa} \wedge \varphi_a \circ \psi_a \sqsubseteq \text{id}_{Ga}$

Composition is of course defined by $(\varphi, \psi) \circ (\varphi', \psi') = (\varphi' \circ \varphi, \psi \circ \psi')$. □

The reason for using $\underline{CPO}_\perp$ instead of $\underline{CPO}$ is that $[\text{Type}, \underline{CPO}]_{PR}$ is not an ω-category, whereas $[\underline{\text{Type}}, \underline{CPO}_\perp]$ is.

THEOREM 19: *$[A, \underline{CPO}_\perp]_{PR}$ is an ω-category for any category A*

Proof: Let A be an arbitrary category. We must show that $[A, \underline{CPO}_\perp]$ has all ω-colimits, *i.e.* that every ω-chain has a colimit, and that $[A, \underline{CPO}_\perp]$ has an initial element.

$\underline{CPO}_\perp$ has all ω-colimits (*see* [LS81]). Then by [HS73] corollary 25.7 $[A, \underline{CPO}_\perp]$ has all ω-colimits, and by [SP82] theorem 2, $(a \Rightarrow e)$, so does $[A, \underline{CPO}_\perp]_{PR}$.

The obvious candidate for an initial object in $[A, \underline{CPO}_\perp]_{PR}$ is the constant functor which maps every A-object to the one-point cpo and every A-morphism to the only possible function between two one-point cpos. It can easily be verified that this is indeed an initial element.

(However, it is not initial in $[A, \underline{CPO}]_{PR}$. Note of the difference between $[A, \underline{CPO}]_{PR}$ and $[A, \underline{CPO}_{PR}]$. The latter is an ω-category (*see* the proof of lemma 2), but only for discrete categories A these two categories are isomorphic.) □

As a consequence of using $\underline{CPO}_\perp$ instead of $\underline{CPO}$, the coercion functions $\text{coerce}_{\sigma\tau}$ have to be strict. (The requirement that the coercions be strict also comes up in [BCGS91], although for different reasons.) From now on we write $\mathscr{K}$ for $[\underline{\text{Type}}, \underline{CPO}_\perp]$. Dom & Coe will be the initial fixed point of the following functor $\mathscr{F}$

DEFINITION 20 ($\mathscr{F} : \mathscr{K}_{\mathscr{P}\mathscr{R}} \to \mathscr{K}_{\mathscr{P}\mathscr{R}}$): $\mathscr{F}$ is a functor $\mathscr{K}_{\mathscr{P}\mathscr{R}}$ to $\mathscr{K}_{\mathscr{P}\mathscr{R}}$, so it consists of an object part, a mapping from $\text{Obj}(\mathscr{K}_{\mathscr{P}\mathscr{R}})$ to $\text{Obj}(\mathscr{K}_{\mathscr{P}\mathscr{R}})$, and an morphism part, a mapping from $\text{Mor}(\mathscr{K}_{\mathscr{P}\mathscr{R}})$ to $\text{Mor}(\mathscr{K}_{\mathscr{P}\mathscr{R}})$.

The object part of $\mathscr{F}$ is defined as follows. Let $F \in \text{Obj}(\mathscr{K}_{\mathscr{P}\mathscr{R}})$. Then $\mathscr{F}\,F \in \text{Obj}(\mathscr{K}_{\mathscr{P}\mathscr{R}})$, *i.e.* $\mathscr{F}\,F$ is a functor from $\underline{\text{Type}}$ to $\underline{CPO}_\perp$.

The functor part of $\mathscr{F}\,F$, a mapping from $\text{Obj}(\underline{\text{Type}})$ to $\text{Obj}(CPO_\perp)$, is defined by (¹)

$$(\mathscr{F}\,F)\,\sigma = \text{domain}_\sigma$$
$$(\mathscr{F}\,F)\,\sigma \to \tau = FS(F\sigma, F\tau)$$
$$(\mathscr{F}\,F)(\Pi\,\alpha : *\,.\,\tau) = GP(\langle\, F(\tau[\alpha := a]) \mid a \in \text{Type}\,\rangle)$$

and the morphism part of $\mathscr{F}\,F$, a mapping from $\text{Mor}(\underline{\text{Type}})$ to $\text{Mor}(\underline{CPO}_\perp)$, is defined by

$$(\mathscr{F}\,F)\,\sigma \leqq \tau = \text{coerce}_{\sigma\tau}$$
$$(\mathscr{F}\,F)\,\sigma \to \tau \leqq \sigma' \to \tau' = FS(F\sigma' \leqq \sigma,\ F\tau \leqq \tau')$$
$$(\mathscr{F}\,F)(\Pi\,\alpha : *\,.\,\sigma) \leqq (\Pi\,\alpha : *\,.\,\tau) = GP(\langle\, F\sigma[\alpha := a] \leqq \tau[\alpha := a] \mid a \in \text{Type}\,\rangle)$$

The morphism part of $\mathscr{F}$ is defined as follows. If $(\varphi, \psi) \in \text{Hom}_{\mathscr{K}_{\mathscr{P}\mathscr{R}}}(F, G)$, so $\varphi : F \dot{\to} G$ and $\psi : G \dot{\to} F$ then $\mathscr{F}((\varphi, \psi)) = (\varphi', \psi')$, *i.e.* $\varphi' : \mathscr{F}\,F \dot{\to} \mathscr{F}\,G$ and $\psi' : \mathscr{F}\,G \dot{\to} \mathscr{F}\,F$ where

$$(\varphi'_\sigma, \psi'_\sigma) = (\text{id}_{\text{domain}_\sigma}, \text{id}_{\text{domain}_\sigma})$$
$$(\varphi'_{\sigma \to \tau}, \psi'_{\sigma \to \tau}) = FS_{PR}((\varphi_\sigma, \psi_\sigma), (\varphi_\tau, \psi_\tau))$$
$$(\varphi'_{\Pi\alpha : *.\tau}, \psi'_{\Pi\alpha : *.\tau}) = GP_{PR}(\langle\, (\varphi_{\tau[\alpha := a]}, \psi_{\tau[\alpha := a]}) \mid a \in \text{Type}\,\rangle)$$

(¹) Of course now $FS : \underline{CPO}_\perp^{op} \times \underline{CPO}_\perp \to \underline{CPO}_\perp$ and $GP : \Pi\,\underline{CPO}_\perp \to \underline{CPO}_\perp$.

Checking $\varphi' : \mathscr{F} F \dot{\rightarrow} \mathscr{F} G$ and $\psi' : \mathscr{F} G \dot{\rightarrow} \mathscr{F} F$ is straightforward, and it can easily be verified (pointwise) that $\mathscr{F}$ preserves identities and composition. □

Note that for the coercions FS is used, which takes care of the contravariance of $\rightarrow$ with respect to the subtype relation, whereas for the morphisms FS_{PR} is used, which is covariant in both arguments, so that a fixed point can be constructed.

Any fixed point of $\mathscr{F}$ will solve the recursive domain equations and satisfy the conditions for the coercion functions. For instance, let $(F, (\Phi, \Psi))$ be a fixed point of $\mathscr{F}$, *i.e.* (Φ, Ψ) is an isomorphism between F and $\mathscr{F} F$. This means that $\Phi : F \dot{\rightarrow} \mathscr{F} F$ and $\Psi : \mathscr{F} F \dot{\rightarrow} F$ such that $\Phi \circ \Psi = \mathrm{id}_{\mathscr{F} F}$ and $\Psi \circ \Phi = \mathrm{id}_F$. Because everything is defined pointwise, this means that for all $a \leqq b$

$$\Phi_b \circ \Psi_b = \mathrm{id}_{(\mathscr{F} F)\, b}$$
$$\Psi_b \circ \Phi_b = \mathrm{id}_{Fb}$$
$$\Phi_a \circ \Psi_a = \mathrm{id}_{(\mathscr{F} F)\, a}$$
$$\Psi_a \circ \Phi_a = \mathrm{id}_{Fa}$$

and

$$\begin{tikzcd}
Fa \arrow[r, shift left, "\Phi_a"] \arrow[d, "F\, a \leqq b"'] \arrow[dr, phantom, "☺"] & (\mathscr{F} F)a \arrow[l, shift left, "\Psi_a"] \arrow[d, "(\mathscr{F} F)\, a \leqq b"] \\
Fb \arrow[r, shift left, "\Phi_b"] & (\mathscr{F} F)b \arrow[l, shift left, "\Psi_b"]
\end{tikzcd}$$

Let $a = (\Pi\, \alpha : * . \sigma)$ and $b = (\Pi\, \alpha : * . \tau)$. Then

$$\begin{aligned}
F(\Pi \alpha : * . \tau) &\leqq (\Pi\, \alpha : * . \sigma) \\
&= \Psi_{\Pi\alpha : * . \tau} \circ ((\mathscr{F} F)(\Pi\, \alpha : * . \sigma) \leqq (\Pi\, \alpha : * . \tau)) \circ \Phi_{\Pi\alpha : * . \sigma} \\
&= \Psi_{\Pi\alpha : * . \tau} \circ GP(\langle F \sigma [\alpha := c] \leqq \tau [\alpha := c] \mid c \in \mathrm{Type} \rangle) \circ \Phi_{\Pi\alpha : * . \sigma}
\end{aligned}$$

So $\mathscr{P}_3$ is satisfied. In the same way it can be shown that $\mathscr{P}_2$ holds.

LEMMA 21: $\mathscr{F}$ *is* ω-*continuous.*

Proof: *See* Appendix B. □

So by the initial fixed point theorem an initial fixed point (Dom & Coe, (Φ, Ψ)) of $\mathscr{F}$ can be constructed. The object part of Dom & Coe gives us the family of cpos Dom, the morphism part gives us the family of coercions Coe, and Φ is the required family of bijections.

So, recapitulating,

- $\underline{CPO_\perp}$ is an O-category;
- $[\underline{\text{Type}}, \underline{CPO_\perp}]$ is an O-category;
- $[\underline{\text{Type}}, \underline{CPO_\perp}]_{PR}$ is an ω-category;
- $\mathscr{F}$ is ω-continuous;
- in $[\underline{\text{Type}}, \underline{CPO_\perp}]_{PR}$ the equation $\mathscr{F}(D) \cong D$ has an initial solution (Dom & Coe, (Φ, Ψ));
- the initial fixed point (Dom & Coe, (Φ, Ψ)) of $\mathscr{F}$ gives us a family of cpos solving the recursive domain equations with the associated bijections, and a family of coercions satisfying the coherence conditions.

5. RECURSIVE TYPES AND SUBTYPING

We now combine the two extensions of λ_2 we have dealt with, subtyping and recursive types. The resulting system is called $\lambda_2\mu\leqq$.

5.1 Syntax

First we consider how to define the subtype relation on recursive types. Subtype judgements are now of the form $C \vdash \sigma \leqq \tau$, where C is a set of type contraints of the form $(\alpha \leqq \beta)$ with α and β type variables. The rule for subtyping on recursive types is

$$\frac{C \cup \{\alpha \leqq \beta\} \vdash \sigma \leqq \tau}{C \vdash (\mu\alpha : *.\sigma) \leqq (\mu\beta : *.\tau)} (\leqq \mu)$$

where $\alpha \notin FV(\tau)$, $\beta \notin FV(\sigma)$ and α and β do not occur in C. A new rule is needed to use the type constraints in this context and, because we now have $\approx$ as type equality, the rule (*REFL*) changes

$$\frac{(\alpha \leqq \beta) \in C}{C \vdash \alpha \leqq \beta} (TC) \qquad \frac{\sigma \approx \tau}{C \vdash \sigma \leqq \tau} (REFL)$$

In all the other rules given in section 4.1 we simply prefix the premises and conclusion by "$C \vdash$".

$$\frac{\sigma \leqq^{B} \tau}{C \vdash \sigma \leqq \tau}(START) \qquad \frac{\sigma : *}{C \vdash \sigma \leqq \sigma}(REFL) \qquad \frac{C \vdash \rho \leqq \sigma \quad C \vdash \sigma \leqq \tau}{C \vdash \rho \leqq \tau}(TRANS)$$

$$\frac{C \vdash \sigma' \leqq \sigma \quad C \vdash \tau \leqq \tau'}{C \vdash \sigma \to \tau \leqq \sigma' \to \tau'}(\leqq \to) \qquad \frac{C \vdash \sigma \leqq \tau}{C \vdash (\Pi\alpha : *.\sigma) \leqq (\Pi\alpha : *.\tau)}(\leqq \Pi)$$

The subsumption rule becomes

$$\frac{\Gamma \vdash M : \sigma \quad \varnothing \vdash \sigma \leqq \tau}{\Gamma \vdash M : \tau}(SUB)$$

i.e. only if $\sigma \leqq \tau$ can be derived without any assumptions may $M : \tau$ be deduced from $M : \sigma$. It is possible to let the type constraints also influence type derivations, resulting in type judgements of the form $C, \Gamma \vdash M : \sigma$. But without bounded quantification this is not interesting, as type variables which occur in type constraints may never be bound and are thus effectively type constants. The rule $(\approx)$ is now just a special case of (SUB). An easy induction on derivations shows that the rule $(TRANS)$ is derivable, as it was for $\lambda_2 \leqq$.

5.2. Semantics: general model definition

Type and $[\![\ ]\!]$ for type expressions are defined as they were for $\lambda_2 \mu$.

The subtype relation on type expressions induces a subtype relation on Type. We extend the notion of covariant and contravariant occurences to trees in the obvious way. So for example, in

$$\begin{tikzcd}[column sep=small] & & \to \arrow[dl, no head] \arrow[dr, no head] & \\ & \to \arrow[dl, no head] \arrow[dr, no head] & & \alpha \\ \text{int} & & \text{real} & \end{tikzcd}$$

int and α occur at covariant positions, whereas real and

$$\begin{tikzcd}[column sep=small] & \to \arrow[dl, no head] \arrow[dr, no head] & \\ \text{int} & & \text{real} \end{tikzcd}$$

occur at contravariant positions.

DEFINITION 22: For s, $t \in$ Type, $s \leqq^{*} t$ iff. except for their leaves, s and t are the same tree, and for all leaves l_s and l_t in the same place in s and t, respectively:

- $l_s \leqq^B l_t$ and l_s and l_t occur at covariant positions in s and t, or
- $l_t \leqq^B l_s$ and l_s and l_t occur at contravariant positions in s and t, or
- $l_s \equiv l_t$ □

We want to prove $\varphi \vdash \sigma \leqq \tau \Leftrightarrow \forall_\eta [\![\sigma]\!]\eta \leqq^* [\![\tau]\!]\eta$. The implication $\Rightarrow$ is the more important one, since if that implication holds, then a family of coercion function $\langle Coe_{ab} \mid a \leqq^* b \rangle$ contains the required coercions.

If η is an environment and C a sets of type contraints we say that $\eta \vDash C$ iff. $\eta(\alpha) \leqq^* \eta(\beta)$ for all $(\alpha \leqq \beta) \in C$.

LEMMA 23: (*Soundness*) $C \vdash \sigma \leqq \tau \Rightarrow \forall_{\eta \vDash C} [\![\sigma]\!]\eta \leqq^* [\![\tau]\!]\eta$.

Proof: Induction on the derivation of $\sigma \leqq \tau$. We only treat the prime case, $(\leqq \mu)$.

Suppose the last rule of the derivation is $(\leqq \mu)$,

$$\frac{C \cup \{\alpha \leqq \beta\} \vdash \sigma \leqq \tau}{C \vdash (\mu\alpha : *.\sigma) \leqq (\mu\beta : *.\tau)}$$

Suppose $\eta \vDash C$. Define

$$F = (\lambda t \in \text{Type}.[\![\sigma]\!]\eta[\alpha := t]) \qquad \text{and} \qquad G = (\lambda t \in \text{Type}.[\![\tau]\!]\eta[\beta := t]).$$

So $[\![(\mu\alpha : *.\sigma)]\!]\eta$ and $[\![(\mu\beta : *.\tau)]\!]\eta$ are the least fixed points of F and G, respectively. By induction on $i \in \mathbb{N}$ we prove $F^i \bot \leqq^* G^i \bot$.

Base: $F^0 \bot = \bot \leqq^* \bot = G^0 \bot$

Step: Let $\eta' = \eta[\alpha := F^i \bot][\beta := G^i \bot]$. By the induction on $i : F^i \bot \leqq^* G^i \bot$, so $\eta' \vDash C \cup \{\alpha \leqq \beta\}$.

By the induction on the derivation $\forall_{\eta \vDash C \cup \{\alpha \leqq \beta\}} [\![\sigma]\!]\eta \leqq^* [\![\tau]\!]\eta$,
so $F^{i+1} \bot = [\![\sigma]\!]\eta' \leqq^* [\![\tau]\!]\eta' = G^{i+1} \bot$.

Now $[\![(\mu\alpha : *.\sigma)]\!]\eta = \sqcup F^i \bot \leqq^* \sqcup G^i \bot = [\![(\mu\alpha : *.\tau)]\!]\eta$. □

DEFINITION 24: (General model definition $\lambda_2\mu\leqq$)

A second order environment model for $\lambda_2\mu\leqq$ is a 4-tuple

$$\langle \text{Dom}, \Phi_{\text{term}}, \mathscr{I}_{\text{term}}, \text{Coe} \rangle,$$

where *Coe* is a family of coercion functions,

$$\text{Coe} = \langle \text{Coe}_{ab} \in [\text{Dom}_a \to \text{Dom}_b] \mid a, b \in \text{Type}, a \leqq^* b \rangle,$$

satisfying $\mathscr{P}_0$, $\mathscr{P}_1$, $\mathscr{P}_2$ and $\mathscr{P}_3$, and the rest as in the definition of the general model definition for $\lambda_2\mu$ (definition 12). □

For the systems $\lambda_2\mu\leqq$ we have the same type inference rules as for $\lambda_2\leqq$. So the proof of coherence for $\lambda_2\leqq$ (theorem 29) also proves coherence for $\lambda_2\mu\leqq$.

5.3. The construction of a cpo model

To construct the required family of cpos and a family of coercion functions the same construction as for $\lambda_2\leqq$ can be used. $\underline{\text{Type}}$ is defined as in definition 17, but now for Type as defined in definition 11 and as the subtype relation $\leqq^*$ as defined in definition 22. We again write $\mathscr{K}$ for $[\underline{\text{Type}}, \underline{CPO_\perp}]$. By Lemma 19, $\mathscr{K}_{PR}$ is again an ω-category.

DEFINITION 25: $[\mathscr{F} : \mathscr{K}_{\mathscr{PR}} \to \mathscr{K}_{\mathscr{PR}}]$

The object part of $\mathscr{F}$ is defined as follows. Let $F \in \text{Obj}(\mathscr{K}_{\mathscr{PR}})$. Then the object part of $\mathscr{F} F$, a mapping from $\text{Obj}(Type)$ to $\text{Obj}(CPO_\perp)$, is defined by

$$(\mathscr{F} F).\sigma = \text{domain}_\sigma$$

$$(\mathscr{F} F)\underset{\sigma\qquad\tau}{\overset{\rightarrow}{/\ \ \backslash}} = FS(F\sigma, F\tau)$$

$$(\mathscr{F} F)\underset{\tau}{\underset{\downarrow}{\Pi_\alpha}} = GP(\langle F(\tau[\alpha := a]) \mid a \in \text{Type}\rangle)$$

$$(\mathscr{F} F)\perp = \text{domain}_\perp$$

and the morphism part of $\mathscr{F} F$, a mapping from $\text{Mor}(\underline{\text{Type}})$ to $\text{Mor}(CPO_\perp)$, is defined by

$$(\mathscr{F} F).\sigma \leqq .\tau = \text{coerce}_{\sigma\tau}$$

$$(\mathscr{F} F)\underset{\sigma\qquad\tau}{\overset{\rightarrow}{/\ \ \backslash}} \leqq \underset{\sigma'\qquad\tau'}{\overset{\rightarrow}{/\ \ \backslash}} = FS(F\sigma' \leqq \sigma, F\tau' \leqq \tau)$$

$$(\mathscr{F} F)\underset{\sigma}{\underset{\downarrow}{\Pi_\alpha}} \leqq \underset{\tau}{\underset{\downarrow}{\Pi_\alpha}} = GP(\langle F\sigma[\alpha := \alpha] \leqq \tau[\alpha := a] \mid a \in \text{Type}\rangle)$$

$$(\mathscr{F} F)\perp \leqq \perp = \text{id}_{\text{domain}_\perp}$$

The morphism part of $\mathcal{F}$ is defined as follows:
if $(\eta, \theta) \in \mathrm{Hom}_{\mathcal{KPR}}(F, G)$, then $\mathcal{F}(\eta, \theta) = (\eta', \theta')$, where

$$(\eta'_{.\sigma}, \theta'_{.\sigma}) = (\mathrm{id}_{\mathrm{domain}_\sigma}, \mathrm{id}_{\mathrm{domain}_\sigma})$$

$$(\eta'_{\substack{\to\\ \swarrow\ \searrow\\ \sigma\quad\tau}}, \theta'_{\substack{\to\\ \swarrow\ \searrow\\ \sigma\quad\tau}}) = FS_{PR}((\eta_\sigma, \theta_\sigma), (\eta_\tau, \theta_\tau))$$

$$(\eta'_{\substack{\Pi_\alpha\\ \downarrow\\ \tau}}, \theta'_{\substack{\Pi_\alpha\\ \downarrow\\ \tau}}) = GP_{PR}(\langle (\eta_{\sigma[\alpha := a]}, \theta_{\sigma[\alpha := a]}) \mid a \in \mathrm{Type} \rangle)$$

$$(\eta'_\perp, \theta'_\perp) = (\mathrm{id}_{\mathrm{domain}_\perp}, \mathrm{id}_{\mathrm{domain}_\perp})$$

In the same way lemma 21 is proved, we can prove that $\mathcal{F}$ is ω-continuous. So $\mathcal{F}$ has an initial fixed point (Dom & Coe, (Φ, Ψ)) which gives us a family of cpos solving the recursive domain equations with the associated bijections, and a family of coercions satisfying the coherence conditions.

6. OTHER EXTENSIONS

To all the systems we described, other type constructors, such as $\times$ (Cartesian product), $+$ (separated sum), $\otimes$ (smashed product), $\oplus$ (coalesced sum) or $(-)_\perp$ (lifting) can easily be added. For the general model definitions the necessary domain equations must be given, and all that is required for the construction of a cpo model is a corresponding functor, like we have the function space functor *FS* for $\to$-types.

Σ-types (or existential types) which can be used for abstract data types (*see* [MP88]), can also be added. These types can be treated like the Π-types. Just like the generalized product functor is used for Π-types, the generalized sum functor (*see* [tEH89*b*]) can be used for Σ-types.

Other interesting extensions are of course labelled products, *i.e.* records, and bounded quantification. We will sketch how these could be incorporated in the model.

For labelled products of the form $\langle l_1 : \sigma_1, \ldots, l_n : \sigma_n \rangle$, where the l_i are distinct labels, the required domain equation is

$$\mathrm{Dom}_{[\![\langle l_1 : \sigma_1, \ldots, l_n : \sigma_n \rangle]\!]\eta} \cong \prod_{l_i \in l_1, \ldots, l_n} \mathrm{Dom}_{[\![\sigma_i]\!]\eta}$$

for which we again use the *GP*-functor

$$\mathrm{Dom}_{[\![\langle l_1:\sigma_1,\ldots,l_n:\sigma_n\rangle]\!]\eta} \cong GP(\langle \mathrm{Dom}_{[\![\sigma_i]\!]\eta} \mid l_i \in \{l_1,\ldots,l_n\}\rangle)$$

The subtyping rule for record-types is

$$\frac{\sigma_1 \leqq \tau_1,\ldots,\sigma_m \leqq \tau_m \quad m \leqq n}{\langle l_1:\sigma_1,\ldots,l_n:\sigma_n\rangle \leqq \langle l_1:\tau_1,\ldots,l_m:\tau_m\rangle}$$

and the associated coherence condition is

$$\begin{aligned}&\mathrm{Coe}_{[\![\sigma]\!]\eta\,[\![\tau]\!]\eta}\\ &= \Phi^{-1}_{[\![\tau]\!]\eta} \circ GP(\langle \mathrm{Coe}_{[\![\sigma_i]\!]\eta\,[\![\tau_i]\!]\eta} \mid l_i \in \{l_1,\ldots,l_m\}\rangle)\\ &\qquad\qquad \circ \langle \mathrm{proj}_{l_i} \mid l_i \in \{l_1,\ldots,l_m\}\rangle \circ \Phi_{[\![\sigma]\!]\eta}\end{aligned}$$

where $\sigma \equiv \langle l_1:\sigma_1,\ldots,l_n:\sigma_n\rangle$, $\tau \equiv \langle l_1:\tau_1,\ldots,l_m:\tau_m\rangle$ and proj_{l_i} is the projection function returning the *i*-th component, so

$$\langle \mathrm{proj}_{l_i} \mid l_i \in \{l_1,\ldots,l_m\}\rangle \in \Big(\prod_{l_i \in \{l_1,\ldots,l_n\}} \mathrm{Dom}_{[\![\sigma_i]\!]\eta}\Big) \to \Big(\prod_{l_i \in \{l_1,\ldots,l_m\}} \mathrm{Dom}_{[\![\sigma_i]\!]\eta}\Big)$$

For coherence we would have to prove that every term still has a minimal type, and that lemma 28 holds for each type derivation rule that is added.

Bounded quantification gives us types of the form $(\Pi\,\alpha \leqq \sigma\,.\,\tau)$. The recursive domain equation for these types is

$$\mathrm{Dom}_{[\![\Pi\alpha\leqq\sigma.\tau]\!]\eta} \cong \prod_{a\in\mathrm{Type},\, a\leqq[\![\sigma]\!]\eta} \mathrm{Dom}_{[\![\tau]\!]\eta[\alpha:=a]}$$

i.e. $\mathrm{Dom}_{[\![\Pi\alpha\leqq\sigma.\tau]\!]\eta} \cong GP(\langle \mathrm{Dom}_{[\![\tau]\!]\eta[\alpha:=a]} \mid a \in \mathrm{Type},\ a \leqq [\![\sigma]\!]\eta\rangle)$

The subtyping rule for Π-types becomes

$$\frac{\alpha \leqq \sigma' \vdash \tau \leqq \tau' \quad \sigma' \leqq \sigma}{(\Pi\,\alpha \leqq \sigma\,.\,\tau) \leqq (\Pi\,\alpha \leqq \sigma'\,.\,\tau')}$$

and for the coercion functions we get the following coherence condition

$$\begin{aligned}&\mathrm{Coe}_{[\![\Pi\alpha\leqq\sigma.\tau]\!]\eta\,[\Pi\alpha\leqq\sigma'.\tau']\!]\eta}\\ &\quad = \Phi^{-1}_{[\![\Pi\alpha\leqq\sigma'.\tau']\!]\eta}\\ &\quad \circ GP(\langle \mathrm{Coe}_{[\![\tau]\!]\eta[\alpha:=a]\,[\![\tau']\!]\eta[\alpha:a]} \mid a \leqq [\![\sigma']\!]\eta >) \circ \langle \mathrm{proj}_a \mid a \in \mathrm{Type}, a \leqq [\![\sigma']\!]\eta\rangle\\ &\quad \circ \Phi_{[\![\Pi\alpha\leqq\sigma.\tau]\!]\eta}\end{aligned}$$

where proj_a is the projection function returning the "a"-th component, so

$$\langle \mathrm{proj}_a \mid a \leqq [\![\sigma']\!]\eta \rangle \in \Big(\prod_{a \leqq [\![\sigma]\!]\eta} \mathrm{Dom}_{[\![\tau]\!]\eta[\alpha:=a]}\Big) \to \Big(\prod_{a \leqq [\![\sigma']\!]\eta} \mathrm{Dom}_{[\![\tau]\!]\eta[\alpha:=a]}\Big)$$

Labelled sums, or variants, and bounded Σ-types can be treated in the same way as labelled products and bounded Π-types. Instead of the generalized product functor *GP* we use the generalized sum functor.

It seems that *F*-bounded quantification [CHC90], developed to capture the notion of inheritance in object-oriented languages, can be modelled in the same way.

7. RELATED WORK AND CONCLUSIONS

In [BL90] and [CG90] coherence is proved for second order lambda calculi extended with bounded quantification and subtyping. In both papers a language is defined without the subsumption rule but with explicit coercion functions instead. A translation is given from type derivations in the original system to terms in the new system, which simply inserts an explicit coercion whenever the subsumption rule is used. Then, assuming that the coercions satisfy certain conditions, it is proved that this translation is coherent, *i.e.* that different type derivations in the original system are mapped to equal terms in the new system. The coherence proof in [BL90] is similar to ours, but requires more coherence conditions. [CG90] prove coherence by defining a normalizing rewriting system on the expressions representing the coercions. For every coherence condition we needed there is a corresponding rewriting rule.

We have not considered a system with explicit coercions as an intermediate step between the original system and a model, as is done in these papers. An advantage of our approach is that fewer coherence conditions are needed, viz. just one for every type constructor, and that the connection between the domain equations and the coherence conditions becomes apparent: for each type constructor there is a corresponding functor, which is used in both the domain equation and the coherence condition. This functor is all that is required for the model construction.

In [BCGS91] it is shown how subtyping in an extended lambda-calculus with bounded quantification can be interpreted via coercion functions that are already definable in the system without subtyping. By introducing constants for the coercions between base types, we could use this technique to model $\lambda_2 \leqq$ using our λ_2-model. This would result in the same interpretation: in the λ_2-model the meaning of the λ_2-term representing the coercion between σ and τ is exactly $\mathrm{Coe}_{[\![\sigma]\!]\eta\,[\![\tau]\!]\eta}$.

The technique described in [BCGS91] does not deal with subtyping between recursive types, so it cannot be used to model $\lambda_2\mu \leqq$ using the $\lambda_2\mu$-model. However, it would seem that the coercions between recursive types are also definable in $\lambda_2\mu$, in the same way it is done in [AC90].

As we have shown, the technique generally used to solve recursive domain equations can be extended to produce not only these domains but also suitable coercions between them. This allows the same fixed-point theorem of [SP82] to be used to construct models for all the systems that we considered. The theory of *O*-categories has proved extremely useful here. Because the functor category $[A, B]$ is an *O*-category if B is, we can use all the standard results for *O*-categories and the associated categories of embedding-projection pairs.

The fact that we have used the category $\underline{CPO}$ is not essential. Other *O*-categories could be used, for instance the category of directed complete partial orderings (posets with lubs of all *directed* sets; a set S is directed if every finite subset of S has an upper bound in S) or complete lattices: types would then be interpreted as directed complete partial orderings or complete lattices.

In [BMM90] the general structure of an environment model for a more powerful second order lambda calculus is given. In this language there are constructor expressions, which, apart from types, can for example be functions from types to types. Type expressions are no longer always in normal form, as they are in λ_2, but can be β and η reduced. In fact, the constructors form a simple typed lamda calculus with a single type-constant "Type" and term-constants "$\rightarrow$" and "Π". A BMM-model contains a submodel for this simply typed lambda calculus.

To use our model construction to make a BMM-model, a term model has to be used for the interpretation of constructors, so types are interpreted as closed type expressions modulo $\beta\eta$. We feel that, when second order lambda calculus is considered as a programming language, such a syntactic interpretation may well be acceptable, because our main interest is then the interpretation of terms and not the computations involving constructors. This model can then be extended to model recursive types and subtyping in the same way we have done with the λ_2-model (*see* [Pol91]).

ACKNOWLEDGEMENTS

We are grateful to H. P. Barendregt, F. Cardone, B. Jacobs and the members of the Eindhoven data type club for discussions on the model construction presented in this paper.

Appendix A: coherence

We now prove that the semantics for $\lambda_2\leqq$ is coherent if the coherence conditions $\mathscr{P}_0$, $\mathscr{P}_1$, $\mathscr{P}_2$ and $\mathscr{P}_3$ hold.

To prove coherence we use the fact that we have *minimal typing* in $\lambda_2\leqq$:

LEMMA 26 (Minimal typing): *In a given context* Γ *every typable term* M *has a minimal type*, i.e. *a type* $\sigma_{\min}$ *such that*

$$\Gamma\vdash M:\sigma_{\min} \qquad \text{and} \qquad \forall\sigma \quad \Gamma\vdash M:\sigma \Rightarrow \sigma_{\min}\leqq\sigma$$

Proof: Induction on the derivation. We only show one case, $(\rightarrow E)$; the others are similar. Suppose we have a derivation of $\Gamma\vdash MN:\tau$ ending with $(\rightarrow E)$.

$$\frac{M:\sigma\rightarrow\tau \quad N:\sigma}{MN:\tau}$$

By the induction hypothesis M and N have a minimal type, say $\rho_{\min}$ and $\sigma_{\min}$, respectively. So $\rho_{\min}\leqq\sigma\rightarrow\tau$ and $\sigma_{\min}\leqq\sigma$. We now prove that $\rho_{\min}$ is an $\rightarrow$-type. By lemma 15 there is a derivation of $\rho_{\min}\leqq\sigma\rightarrow\tau$ that does not use (*TRANS*). This derivation must end with $(\leqq\rightarrow)$ or with (*REFL*), so $\rho_{\min}=\rho_1\rightarrow\rho_2$ for some ρ_1 and ρ_2 with $\sigma\leqq\rho_1$ and $\rho_2\leqq\tau$.

Then $\Gamma\vdash M:\rho_{\min}=\rho_1\rightarrow\rho_2\leqq\sigma\rightarrow\rho_2$, so $\Gamma\vdash MN:\rho_2$. Note that the type ρ_2 does not depend on σ or τ, but only on $\rho_{\min}$.

$\rho_2\leqq\tau$ and τ is an arbitrary type of MN, so ρ_2 is the minimal type of MN. $\square$

Although there may be many type derivations for a term, these type derivations are for a large part determined by the syntax of that term. The problem is that the syntax of a term does not determine if and where the rule (*SUB*) may have been used in a type derivation.

First a few words about notation:

• $[\![\Gamma\vdash M:\sigma]\!]$ is the function $(\lambda\eta\,.\,[\![\Gamma\vdash M:\sigma]\!]\,\eta)$ from environments η that satisfy Γ to $\bigcup_{\eta}\mathrm{Dom}_{[\![\sigma]\!]\,\eta}$.

• Suppose Δ is a derivation deriving $\Gamma \vdash M : \tau$ from

$$\Gamma_1 \vdash N_1 : \sigma_1 \ldots \Gamma_n \vdash N_n : \sigma_n \ ^2$$

i. e.

$$\dfrac{\dfrac{\Gamma_1 \vdash N_1 : \sigma_1}{\vdots} \ \ldots \ \dfrac{\Gamma_n \vdash N_n : \sigma_n}{\vdots}}{\Gamma \vdash M : \tau}$$

Using the definition of $[\![\ ]\!]$, this derivation gives us $[\![\Gamma \vdash M : \tau]\!]$ in terms of $[\![\Gamma \vdash N_1 : \sigma_1]\!] \ldots [\![\Gamma \vdash N_n : \sigma_n]\!]$. In other words, Δ determines a function $\mathscr{R}_\Delta$ such that

$$[\![\Gamma \vdash M : \tau]\!] = \mathscr{R}_\Delta([\![\Gamma \vdash N_1 : \sigma_1]\!] \ldots [\![\Gamma \vdash N_n : \sigma_n]\!])$$

• We write

$$\frac{\Gamma \vdash M : \sigma}{\overline{\Gamma \vdash M : \tau}}$$

for any derivation deriving $\Gamma \vdash M : \sigma$ from $\Gamma \vdash M : \tau$. Such a derivation can only use rule (*SUB*), a number of times

• If (*T*) is a type inference rule, we write

$$\frac{\Gamma_1 \vdash N_1 : \sigma_1 \ldots \Gamma_n \vdash N_n : \sigma_n}{\overline{\Gamma \vdash M : \tau}}(T)$$

if $\Gamma \vdash M : \tau$ can be derived from $\Gamma_1 \vdash N_1 : \sigma_1 \ldots \Gamma_n \vdash N_n : \sigma_n$ using (*T*) exactly once, (*SUB*) any number of times, and no other rules, *i. e.*

$$\dfrac{\dfrac{\Gamma_1 \vdash N_1 : \sigma_1}{\overline{\Gamma_1 \vdash N_1 : ?}} \ \ldots \ \dfrac{\Gamma_n \vdash N_n : \sigma_n}{\overline{\Gamma_n \vdash N_n : ?}}}{\dfrac{\Gamma \vdash M : ?}{\overline{\Gamma \vdash M : \tau}}}(T)$$

(2) We somewhat abuse notation because N_i may be a type, in which case $\sigma_i \equiv *$ and Γ_i is irrelevant.

LEMMA 27: *For all derivations* Δ:

$$\begin{array}{c} \Gamma \vdash M : \sigma \\ \hline\hline \Gamma \vdash M : \tau \end{array}$$

$\mathcal{R}_\Delta$ is the same, viz. $\mathcal{R}_\Delta = \lambda\xi . \mathrm{Coe}_{\sigma\tau} \circ \xi$

Proof: Only the rule (SUB) can have been used, so this follows directly from $\mathcal{P}_0$ and $\mathcal{P}_1$. □

LEMMA 28: *For all type inference rules* (T) *not equal to* (SUB), *all derivations* Δ,

$$\Delta : \quad \begin{array}{c} \Gamma_1 \vdash N_1 : \sigma_1 \ldots \Gamma_n \vdash N_n : \sigma_n \\ \hline\hline \Gamma \vdash M : \tau \end{array} (T)$$

yield the same $\mathcal{R}_\Delta$.

Proof: We distinguish between the four possible choices for $(T) : (\to I)$, $(\to E)$, (ΠI) and (ΠE). For the first two we need $\mathcal{P}_2$, for the last two $\mathcal{P}_3$. We treat only one case, $\to E$; the others are similar. Suppose

$$\Delta : \quad \begin{array}{c} \Gamma \vdash M : \sigma_1 \to \sigma_2 \quad \Gamma \vdash N : \sigma_3 \\ \hline\hline \Gamma \vdash MN : \tau \end{array} (\to E)$$

then there are types ρ_1 and ρ_2 such that $\sigma_3 \leqq \rho_1 \leqq \sigma_1$ and $\sigma_2 \leqq \rho_2 \leqq \tau$ and

$$\Delta : \quad \begin{array}{c} \begin{array}{c} \begin{array}{c} M : \sigma_1 \to \sigma_2 \\ \hline\hline M : \rho_1 \to \rho_2 \end{array} \quad \begin{array}{c} N : \sigma_3 \\ \hline\hline N : \rho_1 \end{array} \\ \hline MN : \rho_2 \end{array} (\to E) \\ \hline\hline MN : \tau \end{array}$$

Using $\mathscr{P}_2$, we can prove that $\mathscr{R}_\Delta$ does not depend on ρ_1 and ρ_2. Throughout the proof we omit Γ and write σ instead of $[\![\sigma]\!]\eta$ for all type expressions.

$$\begin{aligned}
&\Phi_{\rho_1 \to \rho_2}([\![M:\rho_1 \to \rho_2]\!]\eta)\\
&=\Phi_{\rho_1 \to \rho_2}(\mathrm{Coe}_{\sigma_1 \to \sigma_2\, \rho_1 \to \rho_2}[\![M:\sigma_1 \to \sigma_2]\!]\eta), \text{ by lemma 27}\\
&=\Phi_{\rho_1 \to \rho_2}((\Phi^{-1}_{\rho_1 \to \rho_2}\circ FS(\mathrm{Coe}_{\rho_1\,\sigma_1}, \mathrm{Coe}_{\sigma_2\,\rho_2})\circ\Phi_{\sigma_1 \to \sigma_2})[\![M:\sigma_1 \to \sigma_2]\!]\eta), \text{ by } \mathscr{P}_2\\
&=FS(\mathrm{Coe}_{\rho_1\,\sigma_1}, \mathrm{Coe}_{\sigma_2\,\rho_2})(\Phi_{\sigma_1 \to \sigma_2}[\![M:\sigma_1 \to \sigma_2]\!]\eta), \text{ because } \Phi_{\rho_1 \to \rho_2} \text{ is a bijection}\\
&=\mathrm{Coe}_{\sigma_2\,\rho_2}\circ(\Phi_{\sigma_1 \to \sigma_2}[\![M:\sigma_1 \to \sigma_2]\!]\eta)\circ\mathrm{Coe}_{\rho_1\,\sigma_1}, \text{ definition } FS
\end{aligned}$$

and using this we can prove

$$\begin{aligned}
&[\![MN:\tau]\!]\eta\\
&=\mathrm{Coe}_{\rho_2\,\tau}([\![MN:\rho_2]\!]\eta), \text{ by lemma 27}\\
&=\mathrm{Coe}_{\rho_2\,\tau}((\Phi_{\rho_1 \to \rho_2}[\![M:\rho_1 \to \rho_2]\!]\eta)[\![N:\rho_1]\!]\eta), \text{ definition } [\![\;]\!] \text{ for } (\to E)\\
&=\mathrm{Coe}_{\rho_2\,\tau}((\Phi_{\rho_1 \to \rho_2}[\![M:\rho_1 \to \rho_2]\!]\eta)(\mathrm{Coe}_{\sigma_3\,\rho_1}[\![N:\sigma_3]\!]\eta)), \text{ by lemma 27}\\
&=(\mathrm{Coe}_{\rho_2\,\tau}\circ(\Phi_{\rho_1 \to \rho_2}[\![M:\rho_1 \to \rho_2]\!]\eta)\circ\mathrm{Coe}_{\sigma_3\,\rho_1})[\![N:\sigma_3]\!]\eta\\
&=(\mathrm{Coe}_{\rho_2\,\tau}\circ\mathrm{Coe}_{\sigma_2\,\rho_2}\circ(\Phi_{\sigma_1 \to \sigma_2}[\![M:\sigma_1 \to \sigma_2]\!]\eta)\circ\mathrm{Coe}_{\rho_1\,\sigma_1}\circ\mathrm{Coe}_{\sigma_3\,\rho_1})[\![N:\sigma_3]\!]\eta, \text{ see above}\\
&=(\mathrm{Coe}_{\sigma_2\,\tau}\circ(\Phi_{\sigma_1 \to \sigma_2}[\![M:\sigma_1 \to \sigma_2]\!]\eta)\circ\mathrm{Coe}_{\sigma_3\,\sigma_1})[\![N:\sigma_3]\!]\eta, \text{ using } \mathscr{P}_1 \text{ twice}
\end{aligned}$$

So $[\![MN:\tau]\!]=(\lambda\eta\,.\,[\![MN:\tau]\!]\eta)$ does not depend on ρ_1 or ρ_2. □

Lemma 28 and minimal typing are all that is needed to prove coherence:

THEOREM 29 (Coherence): *All derivations of* $\Gamma \vdash M:\tau$ *give the same meaning* $[\![\Gamma \vdash M:\tau]\!]\eta$.

Proof: Induction on the structure of M.

base M is a variable or a constant: trivial.

step Suppose we have two derivations, Δ_1 and Δ_2, for $\Gamma \vdash M:\tau$. Then for both derivations, the last rule other than (SUB) is the same, say (T). Then Δ_1 and Δ_2 are of the following form

$$\Delta_1:\ \dfrac{\dfrac{\dfrac{\Delta_{11}}{\Gamma_i \vdash N_1:\sigma_1}\ \cdots\ \dfrac{\Delta_{1n}}{\Gamma_n \vdash N_n:\sigma_n}}{\Gamma \vdash M:\sigma}(T)}{\Gamma \vdash M:\tau} \qquad \Delta_2:\ \dfrac{\dfrac{\dfrac{\Delta_{21}}{\Gamma_i \vdash N_1:\rho_1}\ \cdots\ \dfrac{\Delta_{2n}}{\Gamma_n \vdash N_n:\rho_n}}{\Gamma \vdash M:\rho}(T)}{\Gamma \vdash M:\tau}$$

By the induction hypothesis, all derivations for $\Gamma_i \vdash N_i : \sigma_i$ yield the same meaning $[\![\Gamma_i \vdash N_i : \sigma_i]\!]$, and the same is true for $\Gamma_i \vdash N_i : \rho_i$. So in Δ_j each Δ_{ji} can be replaced by any other derivation, and the resulting derivation will give the same meaning for $\Gamma \vdash M : \tau$ as Δ_j.

We now use the fact we have minimal typing.

Let α_i be the minimal type of N_i for $i = 1 \ldots n$. Then the following two derivations, Δ_1' and Δ_2', give the same meaning for $\Gamma \vdash M : \tau$ as Δ_1 and Δ_2, respectively:

$$\Delta_1' : \quad \cfrac{\cfrac{\cfrac{\cfrac{\vdots}{\Gamma_1 \vdash N_1 : \alpha_1}}{\Gamma_1 \vdash N_1 : \sigma_1} \quad \cdots \quad \cfrac{\cfrac{\vdots}{\Gamma_n \vdash N_n : \alpha_n}}{\Gamma_n \vdash N_n : \sigma_n}}{\Gamma \vdash M : \sigma}\,(T)}{\Gamma \vdash M : \tau} \qquad \vdots \qquad \Delta_2' : \quad \cfrac{\cfrac{\cfrac{\cfrac{\vdots}{\Gamma_1 \vdash N_1 : \alpha_1}}{\Gamma_1 \vdash N_1 : \rho_1} \quad \cdots \quad \cfrac{\cfrac{\vdots}{\Gamma_n \vdash N_n : \alpha_n}}{\Gamma_n \vdash N_n : \rho_n}}{\Gamma \vdash M : \sigma}\,(T)}{\Gamma \vdash M : \tau}$$

But by lemma 28 for all derivations Δ

$$\Delta : \quad \cfrac{\Gamma_1 \vdash N_1 : \alpha_1 \ldots \Gamma_n \vdash N_n : \alpha_n}{\Gamma \vdash M : \tau}\,(T)$$

$\mathscr{R}_\Delta$ is the same. So Δ_1' and Δ_2' both give the same meaning for $\Gamma \vdash M : \tau$. □

Using the examples on page 238 and 239, we can actually show that the semantics is coherent if and only if $\mathscr{P}_0$, $\mathscr{P}_1$, $\mathscr{P}_2$ and $\mathscr{P}_3$ hold.

Appendix B: ω-continuity of $\mathscr{F}$

To prove that $\mathscr{F}$ as defined in section 4 (definition 20) is ω-continuous we define a bifunctor $\mathscr{H}$ which is contravariant in its first and covariant in its second argument.

DEFINITION 30: $(\mathscr{H} : \mathscr{K}^{OP} \times \mathscr{K} \to \mathscr{K})$

If $(F, G) \in \mathrm{Obj}(\mathscr{K}^{OP} \times \mathscr{K})$, so F and G are functors from Type to $\mathrm{CPO}_\perp$, then $\mathscr{H}(F, G)$ is defined by

$$(\mathscr{H}(F, G))\, a = \mathrm{domain}_a$$

$$(\mathscr{H}(F, G))\, a \to b = FS(F a, G b)$$

$$(\mathscr{H}(F, G))\,(\Pi \alpha : \ast . \tau) = GP(\langle G(\tau[\alpha := a]) \mid a \in \mathrm{Type} \rangle)$$

and

$$(\mathscr{H}(F, G))\, a \leqq b = \mathrm{coerce}_{ab}$$

$$(\mathscr{H}(F, G))\, a \to b \leqq a' \to b' = FS(F a' \leqq a,\ G b \leqq b')$$

$$(\mathscr{H}(F, G))\, (\Pi\alpha : \ast . \sigma) \leqq (\Pi\alpha : \ast . \tau) = GP(\langle G \sigma[\alpha := a] \leqq \tau[\alpha := a] \mid a \in \mathrm{Type} \rangle)$$

If $(\eta, \theta) \in \mathrm{Hom}((F, G), (F', G'))$, so $\eta : F' \dot{\to} F$ and $\theta : G \dot{\to} G'$, then $\mathscr{H}(\eta, \theta)$ is defined by

$$(\mathscr{H}(\eta, \theta))_a = \mathrm{id}_{\mathrm{domain}_a}$$

$$(\mathscr{H}(\eta, \theta))_{a \to b} = FS(\eta_a, \theta_b)$$

$$(\mathscr{H}(\eta, \theta))_{(\Pi\alpha : \ast . \tau)} = GP(\langle \theta_{\tau[\alpha := a]} \mid a \in \mathrm{Type} \rangle)$$

Checking $\mathscr{H}(\eta, \theta) : \mathscr{H}(F, G) \dot{\to} \mathscr{H}(F', G')$ is straightforward, and it can easily be verified (coordinatewise) that $\mathscr{H}$ preserves identities and composition. □

[SP82] describes how a mixed contra/covariant functor like $\mathscr{H}$ can be used to define a functor $\mathscr{H}_{PR}$ which is covariant in both arguments.

$\mathscr{H}_{PR} : \mathscr{K}_{PR} \times \mathscr{K}_{PR} \to \mathscr{K}_{PR}$ is defined by

$$\mathscr{H}_{PR}(F, G) = \mathscr{H}(F, G)$$

$$\mathscr{H}_{PR}((\eta, \theta), (\varphi, \psi)) = (\mathscr{H}(\theta, \varphi), \mathscr{H}(\eta, \psi))$$

LEMMA 31: *$\mathscr{H}_{PR}$ is ω-continuous.*

Proof: First we prove that $\mathscr{H}$ is *locally continuous*, *i.e.* that $\mathscr{H}$ is ω-continuous when viewed as a map from hom-sets to hom-sets. Because the ordering on the hom-sets of $\mathscr{K}$ is defined coordinatewise, we can prove this coordinatewise.

Let $\langle (\eta^i, \theta^i) \rangle_{i \in \mathbb{N}}$ be an ascending chain in $\mathrm{Hom}((F, G), (F', G'))$, so $\eta^i : F' \dot{\to} F$, $\theta^i : G \dot{\to} G'$, $\eta^i \sqsubseteq \eta^{i+1}$ and $\theta^i \sqsubseteq \theta^{i+1}$.

We must prove

$$\sqcup \mathscr{H}(\eta^i, \theta^i) = \mathscr{H}(\sqcup \eta^i, \sqcup \theta^i)$$

which is equivalent to

$$\forall_{a \in \mathrm{Type}} (\sqcup \mathscr{H}(\eta^i, \theta^i))_a = (\mathscr{H}(\sqcup \eta^i, \sqcup \theta^i))_a$$

because lubs are taken pointwise.

We distinguish three cases: a is a base type, a is a function type, and a is a polymorphic type. For base types it is trivial. For function types it follows from local continuity of FS, and for polymorphic types it follows from local continuity of GP:

$\rightarrow$-types:

$$\begin{aligned}&(\sqcup \mathscr{H}(\eta^i, \theta^i))_{a \to b}\\ &= \sqcup FS(\eta^i_a, \theta^i_b)\\ &= FS(\sqcup \eta^i_a, \sqcup \theta^i_b)\\ &= (\mathscr{H}(\sqcup \eta^i, \sqcup \theta^i))_{a \to b}\end{aligned}$$

Π-types:

$$\begin{aligned}&(\sqcup \mathscr{H}(\eta^i, \theta^i))_{\Pi\alpha:*.\tau}\\ &= \sqcup GP(\langle \theta^i_{\tau[\alpha:=a]} \mid a \in \mathrm{Type} \rangle)\\ &= GP(\langle \sqcup \theta^i_{\tau[\alpha:=a]} \mid a \in \mathrm{Type} \rangle)\\ &= (\mathscr{H}(\sqcup \eta^i, \sqcup \theta^i))_{\Pi\alpha:*.\tau}\end{aligned}$$

So $\mathscr{H}$ is locally continuous.

$\underline{CPO_\perp}$ has all ω-colimits (*see* [LS81]). So by [HS73] corollary 25.7 $\mathscr{K} = [\underline{\mathrm{Type}}, \underline{CPO_\perp}]$ also has all ω-colimits. Then by [SP82], corollary to theorem 2, $\mathscr{K}$ has locally determined colimits of embeddings, and so we may use [SP82] theorem 3, to conclude that $\mathscr{H}_{PR} : (\mathscr{K}^{OP} \times \mathscr{K})_{PR} \to \mathscr{K}_{PR}$ is ω-continuous from the fact that $\mathscr{H}$ is locally continuous. □

LEMMA 32: *$\mathscr{F}$ is ω-continuous.*

Proof: We have the following correspondence between $\mathscr{F}$ and $\mathscr{H}_{PR}$

$$\mathscr{F} F = \mathscr{H}_{PR}(F, F)$$

$$\mathscr{F}(\eta, \theta) = (\mathscr{H}(\eta, \theta), \mathscr{H}(\eta, \theta))$$

So we can get $\mathscr{F}$ by composing $\mathscr{H}_{PR}$ with a diagonal functor from $\mathscr{K}_{PR}$ to $\mathscr{K}_{PR} \times \mathscr{K}_{PR}$. $\mathscr{H}_{PR}$ is ω-continuous, hence so is $\mathscr{F}$. □

REFERENCES

[ABL86] R. AMADIO, K. B. BRUCE and G. LONGO, The finitary projection model for second order lambda calculus and higher order domain equations, *Logic in Computer Science*, 1986, pp. 122-135, IEEE.

[AC90] R. M. AMADIO and L. CARDELLI, *Subtyping recursive types*, Technical Report 62, Digital Systems Research Centre, 1990.

[Bar9+] H. P. BARENDREGT, Typed lambda calculi. In D. M. GABBAI, S. ABRAMSKY and T. S. E. MAIBAUM, Eds., *Handbook of Logic in Computer Science*, volume 1. Oxford University Press, to appear.

[BH88] R. Bos and C. Hemerik, *An introduction to the category-theoretic solution of recursive domain equations*, Technical Report 15, Eindhoven University of Technology, 1988.

[BL90] K. B. Bruce and G. Longo, A modest model of records, iheritance and bounded quantification, *Information and Computation*, 1990, *87*, pp. 196-240.

[BMM90] K. B. Bruce, A. R. Meyer and J. C. Mitchell, The semantics of second-order lambda calculus, *Information and Computation*, 1990, *85*, pp. 76-134.

[BCGS91] V. Breazu-Tannen, Th. Coquand, C. A. Gunter and A. Scedrov. Inheritance as explicit coercion. *Information and Computation*, 1991, *93*, (1), pp. 172-221.

[CC91] F. Cardone and M. Coppo, Type inference with recursive types: Syntax and semantics, *Information and Computation*, 1991, *92*, (1), pp. 48-80.

[CG90] P.-L. Curien and G. Ghelli, Coherence of subsumption. In A. Arnold, Ed., *Colloquium on Trees in Algebras and Programming*, Vol. 431 of LNCS, 1990, pp. 132-146, Springer.

[CHC90] W. R. Cook, W. L. Hill and P. S. Canning, Inheritance is not subtyping, *Principles of Programming Languages*, 1990, pp. 125-135, ACM.

[CL90] L. Cardelli and G. Longo, *A semantic basis for Quest*, Technical Report 55, Digital Systems Research Center, Palo Alto, California 94301, 1990.

[CM89] L. Cardelli and J. C. Mitchell, Operations on records, in M. Main *et al.*, Ed., *Fifth International Conference on Mathematical Foundations of Programming Semantics*, Vol. 442 of LNCS, 1989, pp. 22-53.

[Cou83] B. Courcelle, Fundamental properties of infinite trees, *Theoretical Computer Science*, 1983, *25*, pp. 95-169.

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

[Gir72] J.-Y. Girard, *Interprétation fonctionnelle et élimination des coupures de l'arithmétique d'ordre supérieur*, Ph. D. thesis, Université Paris-VII, 1972.

[Gir86] J.-Y. Girard, The system *F* of variable types, fifteen years later. *Theoretical Computer Science*, 1986, *45*, pp. 159-192.

[HS73] H. Herrlich and G. E. Strecker. *Category Theory*. Allyn and Bacon, 1973.

[LS81] D. J. Lehmann and M. B. Smyth, Algebraic specification of data types: a synthetic approach, *Math. Syst. Theory*, 1981, *11*, pp. 97-139.

[McC79] N. McCracken, *An Investigation of a Programming Language with a Polymorphic Type Structure*, Ph. D. thesis, Syracuse University New York, 1979.

[Mit84] J. C. Mitchell, Semantic models for second-order lambda calculus, *Foundations of Computer Science*, 1984, pp. 289-299, IEEE.

[MP88] J. C. Mitchell and G. D. Plotkin, Abstract types have existential type, *ACM Trans. on Prog. Lang. and Syst.*, 1988, *10*, (3), pp. 470-502.

[Pol91] E. Poll, *Cpo-models for second order lambda calculus with recursive types and subtyping*, Computing Science Note (91/07), Eindhoven University of Technology, 1991.

[Rey74] J. C. Reynolds, Towards a theory of type structure, *Programming Symposium: Colloque sur la Programmation*, LNCS, 1974, pp. 408-425, Springer.

[SP82] J. C. Smyth and G. D. Plotkin, The category-theoretic solution of recursive domain equations, *S.I.A.M. Journal of Computing*, 1982, *11*, pp. 761-783.

[tEH89a] H. Ten Eikelder and C. Hemerik, The construction of a cpo model for second order lambda calculus with recursion, *Procs. CNS'89 Computing Science in the Netherlands*, 1989, pp. 131-148.

[tEH89b] H. Ten Eikelder and C. Hemerik, *Some category-theoretical properties related to a model for a polymorphic lambda calculus*, Computing Science Note (89/03), Eindhoven University of Technology, 1989.