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

Eindhoven University of Technology

Department of Mathematics and Computing Science

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

by

Erik Poll

Computing Science Note 91/07
Eindhoven, September 1991

COMPUTING SCIENCE NOTES

Copies can be ordered from:
Mrs. F. van Neerven
Eindhoven University of Technology
Department of Mathematics and Computing Science
P.O. Box 513
5600 MB EINDHOVEN
The Netherlands
ISSN 0926-4515

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

Erik Poll *

Abstract

In this paper we present constructions of cpo models for second order lambda calculi with recursive types and/or subtyping. The model constructions are based on a model construction by ten Eikelder and Hemerik for second order lambda calculus with recursive types ([tEH89a]). The models will be 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.

*supported by the Dutch organization for scientific research (NWO).

Contents

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.

In this paper we consider two extensions of the second order lambda calculus: subtyping and recursive types. We first construct a model for the second order lambda calculus, and then show how this construction can be adapted to include subtyping and recursive types.

Both subtyping and recursive types are interesting from the point of view of programming languages. Recursive types can be used to make types such as list and trees. Also fixed point operators, which cannot be typed in second order lambda calculus, can be typed using recursive types.
Subtyping can also be found in progamming languages: in combination with labelled records it 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].

We only consider a very simple form of subtyping. We do not have labelled records or bounded quantification, as for instance in *Fun*, but instead all subtyping will be based on a subtype relation on a set of base types. For example, if we have base types *int* and *real* we could have $int \leq real$, i.e. *int* is a subtype of *real*. In the final section we will show that the incorporation of bounded quantification and record types in the models is straightforward.

Several models for second order lambda calculus are known, for example models based on partial equivalence relations [Gir72], the closure model [Mac79], the finitary projection model [ABL86] and models based on qualitative domains [Gir86].

The model constructions in this paper are based on a model construction by ten Eikelder and Hemerik for second order lambda calculus with recursive types [tEH89a]. The models 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.
A pleasing aspect of the model constructions is that other type constructors, such as Σ (existential types), $\times$ (Cartesian product) , $+$ (seperated sum), $\otimes$ (smashed product) , $\oplus$ (coalesced sum) or $(\_)_\perp$ (lifting) can easily be added.

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. We must prove that all derivations for a term give the same meaning, which is called *coherence*. Examples of coherence proofs can be found in [BTCGS89] and [CG90]. In both papers coercions are used to interpret a second order λ-calculus with subtyping, and coherence is proved for this interpretation.

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* [BL88] 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 [BTCGS89] 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 models 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 variables.

Before we combine subtyping and recursive types, we first consider them separately. We will consider several ways to define equality for recursive types, each resulting in slightly different systems.
For all resulting systems we give general model definitions similar to the definition of a Bruce-Meyer-Mitchell environment model [BMM90], and we construct cpo models based on those general model definitions. An advantage of the general model definitions is that we can prove properties not just for one particular model but for all models that fit the general model definition. For example, for the systems with subtyping we can prove that a model is coherent, if the coercions satisfy certain conditions.

Once we have the general model definition, the construction of a model is relatively simple.
For the systems without subtyping, the model constructions are slight modifications of the one 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, the problem of the contravariance of $\sigma \rightarrow \tau$ in σ can be overcome in the standard way, by working in a category of embedding-projection pairs, a technique described in [SP82][BH88]. A solution for the recursive domain equation is then found using a standard fixed point construction for ω-continuous functors on a suitable product category of $\underline{CPO}_{PR}$ (an inverse-limit construction).
For the systems with subtyping, 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 a category corresponding with the subtype relation on types to $\underline{CPO}$. Such a functor, satifying both the recursive domain equations and the coherence conditions, is again found by an inverse limit construction, only this time in a functor category. The problem of the contravariance of $\rightarrow$ is overcome in the same way as for the systems without subtyping, viz. by using projection-embedding pairs. For the rather technical proofs of the categorical properties needed for this construction we refer to [Pol91].

In the following section we give a short description of the second order lambda calculus. The version we describe is identical to the one described in [BMM90],[Mit84] and [ABL86]. We then give the definition of a Bruce-Meyer-Mitchell environment model and construct a cpo model based on that definition.
In section 3 we consider several ways to extend second order lambda calculus with recursive types. For each possibility we give a general model definition and we construct a cpo model.
In section 4 we then describe the second order lambda calculus extended with subtyping, and again we give a general model definition and construct a cpo model, and in section 5 we consider the second order lambda calculus with both subtyping and recursive types.

Finally in the concluding section we indicate how bounded quantification , record types and other extensions can be included in the model, and we briefly discuss the model constructions.

2 Second order lambda calculus

2.1 Syntax

We will now give a short description of the second order lambda calculus (Λ for short). The system described here is the same as in [BMM90], [ABL86] and [Mit84].

We distinguish three sorts of expressions : kinds, constructors and terms.
Every term has a type. Types are made using constructors. In fact, the types themselves are also constructors. The constructors also have "types", which we call kinds.

kinds

The set of kinds is given by

$$\kappa ::= * \mid \kappa_1 \Rightarrow \kappa_2 \quad .$$

Kinds are the "types" of construction expressions.

constructors and their kinds

Let $\mathcal{C}_{cons}$ be a set of constructor constants and $\mathcal{V}_{cons}$ be a set of constructor variables. All constructors constants have a specified kind, which we will write as a superscript when necessary. First we define the set of *pseudo-constructors* over $\mathcal{C}_{cons}$ and $\mathcal{V}_{cons}$, of which the set of constructor expressions will be a subset.
The set of pseudo-constructors over $\mathcal{C}_{cons}$ and $\mathcal{V}_{cons}$ is given by:

$$\sigma = c \mid \alpha \mid \sigma_1\sigma_2 \mid (\Lambda\alpha : \kappa.\sigma)$$

where $c \in \mathcal{C}_{cons}$, $\alpha \in \mathcal{V}_{cons}$ and κ a kind.
The system Λ we describe here is not quite the same as Girard's system F or $\lambda 2$ in Barendregt's cube [Bar9], because we allow abstraction over all kinds here and not just over types. In the terms however, we shall only allow abstraction over types.
Constructors are those pseudo-constructors for which a kind can be derived in a context. A context here is a syntactic kind assignment, i.e. a partial function from $\mathcal{V}_{cons}$ to the set of kinds. So a context assigns kinds to constructor variables. We write $\Gamma \vdash \sigma : \kappa$ if we can derive that in context Γ the constructor σ has kind κ, using the following rules:

$$\Gamma \vdash c^\kappa : \kappa \qquad (c^\kappa \in \mathcal{C}_{cons}) \qquad \Gamma, \alpha : \kappa \vdash \alpha : \kappa \qquad (\alpha \in \mathcal{V}_{cons})$$

$$\frac{\Gamma, \alpha : \kappa_1 \vdash \sigma : \kappa_2}{\Gamma \vdash (\Lambda\alpha : \kappa_1.\sigma) : \kappa_1 \Rightarrow \kappa_2}(\Rightarrow I) \qquad \frac{\Gamma \vdash \sigma : \kappa_1 \Rightarrow \kappa_2 \qquad \Gamma \vdash \tau : \kappa_1}{\Gamma \vdash \sigma\tau : \kappa_2}(\Rightarrow E)$$

Constructor expressions of kind $*$ will be called *type* expressions. $\Gamma \vdash \sigma : *$ means that σ is a type in context Γ.
We assume that $\mathcal{C}_{cons}$ contains the following constants:

$$\begin{array}{lcll} \rightarrow & : & * \Rightarrow (* \Rightarrow *) & \text{(for function types)} \\ \Pi & : & (* \Rightarrow *) \Rightarrow * & \text{(for polymorphic types)} \end{array}$$

We also have constructor constants of kind $*$, which we call the the base types. For example, these might include the types *bool*, *int* or *real*.

So, for example

$$\begin{aligned} \alpha : * &\vdash \ \rightarrow \alpha : * \Rightarrow * \\ \alpha : * &\vdash \ \rightarrow \alpha\alpha : * \end{aligned}$$

$\rightarrow$ will be written infix.

$$\begin{aligned} \alpha : * &\vdash \ \alpha \rightarrow \alpha : * \\ <> &\vdash \ (\Lambda\alpha : *.\alpha \rightarrow \alpha) : * \Rightarrow * \\ <> &\vdash \ \Pi(\Lambda\alpha : *.\alpha \rightarrow \alpha) : * \end{aligned}$$

The constructor expressions form a simple typed lambda calculus. Equality on constructor expresions is $\beta\eta$-equality. If in a context Γ constructors σ and τ are equal, we write $\Gamma \vdash \sigma =_c \tau$.
The following are well-known properties of simple typed λ-calculus.

1 property

(i) the kind of a constructor in a given context is unique

(ii) equal constructors have the same kind

□

terms and their types

We will now define the set of term expressions, in the same way as we defined the set of constructor expressions.

Let $\mathcal{C}_{term}$ be a set of term constants and $\mathcal{V}_{term}$ be a set of term variables. 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 $\mathcal{C}_{term}$ and $\mathcal{V}_{term}$, of which the set of term expressions will be a subset.
The set of pseudo-terms over $\mathcal{C}_{term}$ and $\mathcal{V}_{term}$ is given by:

$$M = c \mid x \mid (\lambda x : \sigma.M) \mid M_1M_2 \mid (\Lambda\alpha : *.M) \mid M\sigma$$

where $x \in \mathcal{V}_{term}$, $c \in \mathcal{C}_{term}$, $\alpha \in \mathcal{V}_{cons}$ and σ a pseudo-constructor.
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_1M_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. We extend the notion of a context to a partial function on $\mathcal{V}_{cons} \cup \mathcal{V}_{term}$, which assigns kinds to constructor variables and types to term variables.
We write $\Gamma \vdash M : \sigma$ if we can derive that in context Γ the term M has type σ, using the following rules:

$$\Gamma \vdash c^\sigma : \sigma \quad (c^\sigma \in \mathcal{C}_{term}) \qquad \Gamma, x : \sigma \vdash x : \sigma \quad (x \in \mathcal{V}_{term})$$

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

$$\frac{\Gamma, \alpha : * \vdash M : \tau}{\Gamma \vdash (\Lambda\alpha : *.M) : \Pi(\Lambda\alpha : *.\tau)} \; (\Pi\ I) \qquad \frac{\Gamma \vdash M : \Pi f \quad \Gamma \vdash \sigma : *}{\Gamma \vdash M\sigma : f\sigma} \; (\Pi\ E)$$

$$\frac{\Gamma \vdash M : \sigma \quad \Gamma \vdash \sigma =_c \tau}{\Gamma \vdash M : \tau} \; (TEQ)$$

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

2 property

(i) the type of a term in a given context is unique (up to $\beta\eta$-equality)

(ii) equal terms have equal types

□

2.2 Semantics : general model definition

We now give the general structure of an environment model for second order lambda calculus, as described in [BMM90]. The difference is that types are interpreted as cpos whereas in [BMM90] types are interpreted as sets. Because terms may depend on types (in terms $M\sigma$), but types and other constructors cannot depend on terms, we can first consider the semantics of constructor expressions seperately.

the semantics of constructor expressions

As we mentioned earlier, constructors (with their kinds) form a simple typed typed lambda calculus. So as the (sub)model for the constructor expressions we can take a model for the simple type lambda calculus.

3 definition (environment model for constructor expressions)
An environment model for the constuctor expressions over $\mathcal{V}_{cons}$ and $\mathcal{C}_{cons}$ is a 3-tuple $< Kind, \Phi_{cons}, \mathcal{I}_{cons} >$, where

- $Kind =< Kind_\kappa \mid \kappa$ is a kind $>$ is a family of sets, indexed by kind expressions.
- $\Phi_{cons} =< \Phi_{\kappa_1 \Rightarrow \kappa_2} \mid \kappa_1 \Rightarrow \kappa_2$ is a kind $>$ is a family of bijections such that $\Phi_{\kappa_1 \Rightarrow \kappa_2} \in Kind_{\kappa_1 \Rightarrow \kappa_2} \longrightarrow [Kind_{\kappa_1} \rightarrow Kind_{\kappa_2}]$,
  where the square brackets denote some subset of the function space.
- $\mathcal{I}_{cons} \in \mathcal{C}_{cons} \rightarrow \bigcup_\kappa Kind_\kappa$ gives the meanings of the constructor constants. Of course $\mathcal{I}_{cons}(c^\kappa) \in Kind_\kappa$ for all $c^\kappa \in \mathcal{C}_{cons}$.

□

The meaning of a constructor expression of kind κ will be a element of the set $Kind_\kappa$.
The bijections $\Phi_{\kappa_1 \Rightarrow \kappa_2}$ are the element-to-function mappings, well-known from models of the type-free lambda calculus. In fact, for the simple typed lambda calculus we do not need the $\Phi_{\kappa_1 \Rightarrow \kappa_2}$; we can take $Kind_{\kappa_1 \Rightarrow \kappa_2} = Kind_{\kappa_1} \rightarrow Kind_{\kappa_2}$ and all the $\Phi_{\kappa_1 \Rightarrow \kappa_2}$ the identity on $Kind_{\kappa_1 \Rightarrow \kappa_2}$.

We maintain the $\Phi_{\kappa_1 \Rightarrow \kappa_2}$ here to emphasize the similarity with the definition of the semantics of terms that will be given later.

If we can derive $\Gamma \vdash \sigma : \kappa$, $[\![\Gamma \vdash \sigma : \kappa]\!] \eta$ is the meaning of the constructor expressions σ in environment η. Here an environment η is a function which gives the meanings of the free constructor variables occurring in Γ, so $\eta \in \mathcal{V}_{cons} \rightarrow \bigcup_\kappa Kind_\kappa$.

We say that environment η satisfies context Γ, written $\Gamma \models \eta$, if $\eta(\alpha) \in Kind_\kappa$ for all $\alpha : \kappa$ in Γ. For these environments we define the semantics of constructor expressions, by induction on their kind derivation, as follows:

$$\begin{aligned}
[\![\Gamma \vdash \alpha : \kappa]\!] \eta &= \eta(\alpha) && (1)\\
[\![\Gamma \vdash c : \kappa]\!] \eta &= \mathcal{I}_{cons}(c) && (2)\\
[\![\Gamma \vdash \sigma\tau : \kappa_2]\!] \eta &= (\Phi_{\kappa_1 \Rightarrow \kappa_2} [\![\Gamma \vdash \sigma : \kappa_1 \Rightarrow \kappa_2]\!] \eta) \quad [\![\Gamma \vdash \tau : \kappa_2]\!] \eta && (3)\\
[\![\Gamma \vdash (\Lambda\alpha : \kappa_1 . \sigma) : \kappa_1 \Rightarrow \kappa_2]\!] \eta &= \Phi^{-1}_{\kappa_1 \Rightarrow \kappa_2}(\lambda a \in Kind_{\kappa_1} . [\![\Gamma, \alpha : \kappa_1 \vdash \sigma : \kappa_2]\!] \eta[\alpha := a]) && (4)
\end{aligned}$$

Remember that every constructor has a unique kind, so there is only one possible choice for the kind κ_1 of σ in (3). This guarantees that (3) defines a unique meaning for $\sigma\tau$.

For the semantics of construction expression to be defined correctly

$$\Phi^{-1}_{\kappa_1 \Rightarrow \kappa_2}(\lambda a \in Kind_{\kappa_1} . [\![\Gamma, \alpha : \kappa_1 \vdash \sigma : \kappa_2]\!] \eta[\alpha := a])$$

has to be defined for all possible Γ and σ. In other words, the range of the $\Phi_{\kappa_1 \Rightarrow \kappa_2}$ must be large enough. In the actual models we will construct this will never be a problem. We will always have $Kind_{\kappa_1 \Rightarrow \kappa_2} = Kind_{\kappa_1} \rightarrow Kind_{\kappa_2}$, and $\Phi_{\kappa_1 \Rightarrow \kappa_2}$ the identity on $Kind_{\kappa_1 \Rightarrow \kappa_2}$.

For this definition of a constructor model kind we can prove soundness,

$$[\![\Gamma \vdash \rho : \kappa]\!] \eta \in Kind_\kappa \quad ,$$

as well as soundness with respect to constructor equality,

$$\Gamma \vdash \sigma =_c \tau : \kappa \Rightarrow [\![\Gamma \vdash \sigma : \kappa]\!] \eta = [\![\Gamma \vdash \tau : \kappa]\!] \eta$$

(see [BMM90]).

the semantics of terms

The definition of the semantics of terms will be similar to the definition of the semantics of constructors.

Instead of having a family of sets $Kind$, indexed by kinds, we will now need a family of cpos Dom, indexed by $Kind_*$. As for the constructor expressions, we can only talk about the meaning of terms in a context and a matching environment. The meaning of $\Gamma \vdash M : \sigma$ in an environment η will be an element of $Dom_{[\![\Gamma \vdash \sigma : *]\!] \eta}$.

To define the semantics of terms we will need mappings similar to the element-to-function mappings $\Phi_{\kappa_1 \Rightarrow \kappa_2}$ we needed to define the semantics of constructors. However, because we have two kinds of abstraction, over term and over type variables, it will be slightly more complicated.

First we consider the function types.

Suppose $\Gamma \vdash M : \sigma \rightarrow \tau$. Then for all $\Gamma \vdash N : \sigma$ we have $\Gamma \vdash MN : \tau$, so we should be able to define the meaning of MN ($\in Dom_{[\![\Gamma \vdash \tau : *]\!] \eta}$) in terms of the meanings of M ($\in Dom_{[\![\Gamma \vdash \sigma \rightarrow \tau : *]\!] \eta}$)

and N ($\in Dom_{[\Gamma\vdash\sigma:*]\eta}$). To get the meaning of MN, the meaning of M has to be considered as a mapping from $Dom_{[\Gamma\vdash\sigma:*]\eta}$ to $Dom_{[\Gamma\vdash\tau:*]\eta}$. So we require

$$Dom_{[\Gamma\vdash\sigma\to\tau:*]\eta} \cong [Dom_{[\Gamma\vdash\sigma:*]\eta} \to Dom_{[\Gamma\vdash\tau:*]\eta}] \tag{i}$$

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

$$\Phi_{[\Gamma\vdash\sigma\to\tau:*]\eta} \in Dom_{[\Gamma\vdash\sigma\to\tau:*]\eta} \longrightarrow [Dom_{[\Gamma\vdash\sigma:*]\eta} \to Dom_{[\Gamma\vdash\tau:*]\eta}]$$

is the element-to-function mapping that we need to define the meaning of term abstraction and application.

For polymorphic types we need different mapppings.
Suppose $\Gamma \vdash M : \Pi f$. Then for all τ, $\Gamma \vdash \tau : *$, we have $\Gamma \vdash M\tau : f\tau$. So we should be able to define the meaning of $M\tau \in Dom_{[\Gamma\vdash f\tau:*]\eta}$ in terms of the meanings of M and τ , which are elements of $Dom_{[\Gamma\vdash\Pi f:*]\eta}$ and $Kind_*$, respectively. This is achieved by requiring

$$Dom_{[\Gamma\vdash\Pi f:*]\eta} \cong \prod_{a\in Kind_*} Dom_{[\Gamma,\alpha:*\vdash f\alpha:*]\eta[\alpha:=a]} \tag{ii}$$

where α is of course a fresh type variable.
The isomorphism corresponding with (ii), the bijection

$$\Phi_{[\Gamma\vdash\Pi f:*]\eta} \in Dom_{[\Gamma\vdash\Pi f:*]\eta} \longrightarrow \prod_{a\in Kind_*} Dom_{[\Gamma,\alpha:*\vdash f\alpha:*]\eta[\alpha:=a]}$$

will be used to define the meaning of type abstraction and application.

We now have recursive domain equations for all function types and all polymorphic types. For the sake of a more uniform treatment, we also want a recursive 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 will require

$$Dom_\sigma \cong domain_\sigma \tag{iii}$$

For all $a \in Kind_*$, we define a function F_a that maps a family of cpos to a single cpo.
If $< D_a \mid a \in Kind_* >$ is a family of cpos, then

$$\begin{aligned}
F_{[\Gamma\vdash\sigma:*]\eta} < D_a \mid a \in Kind_* >) &= domain_\sigma \qquad \text{for base types } \sigma\\
F_{[\Gamma\vdash\sigma\to\tau:*]\eta}(< D_a \mid a \in Kind_* >) &= [D_{[\Gamma\vdash\sigma:*]\eta} \longrightarrow D_{[\Gamma\vdash\tau:*]\eta}]\\
F_{[\Gamma\vdash\Pi f:*]\eta}(< D_a \mid a \in Kind_* >) &= \textstyle\prod_{a\in Kind_*} D_{[\Gamma,\alpha:*\vdash f\alpha:*]\eta[\alpha:=a]}
\end{aligned}$$

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

$$\forall_{\, a \in Kind_*} : \quad Dom_a \cong F_a(Dom)$$

We will now give the general model definition for second order environment models.

4 definition (second order environment model)
A second order environment model for Λ over $\mathcal{V}_{term}$, $\mathcal{C}_{term}$, $\mathcal{V}_{cons}$ and $\mathcal{C}_{cons}$ is a 6-tuple
$< Kind, \Phi_{cons}, \mathcal{I}_{cons}, Dom, \Phi_{term}, \mathcal{I}_{term} >$,
where

- $< Kind, \Phi_{cons}, \mathcal{I}_{cons} >$ is an environment model for the constructor expressions over $\mathcal{V}_{cons}$ and $\mathcal{C}_{cons}$.
- $Dom = < Dom_a \mid a \in Kind_* >$ is a family of cpos.
- $\Phi_{term} = < \Phi_a \mid a \in Kind_* >$ is a family of continuous bijections such that

$$\Phi_a \in Dom_a \longrightarrow F_a(Dom)$$

  where the F_a are defined by

$$\begin{aligned}
F_{[\![\Gamma \vdash \sigma:*]\!]\eta} < D_a \mid a \in Kind_* >) &= domain_\sigma \qquad \text{for base types } \sigma \\
F_{[\![\Gamma \vdash \sigma \rightarrow \tau:*]\!]\eta}(< D_a \mid a \in Kind_* >) &= [D_{[\![\Gamma \vdash \sigma:*]\!]\eta} \longrightarrow D_{[\![\Gamma \vdash \tau:*]\!]\eta}] \\
F_{[\![\Gamma \vdash \Pi f:*]\!]\eta}(< D_a \mid a \in Kind_* >) &= \prod_{a \in Kind_*} D_{[\![\Gamma, \alpha:* \vdash f\alpha:*]\!]\eta[\alpha:=a]}
\end{aligned}$$

- $\mathcal{I}_{term} \in \mathcal{C}_{term} \rightarrow \bigcup_{a \in Kind_*} Dom_a$ gives the meanings of the term constants. Of course $\mathcal{I}_{term}(c^\sigma) \in Dom_{[\![\Gamma \vdash \sigma:*]\!]\eta}$ for all $c^\sigma \in \mathcal{C}_{term}$.

□

If we can derive $\Gamma \vdash M : \sigma$, $[\![\Gamma \vdash M : \sigma]\!]\eta$ is the meaning of M with type σ in environment η. It will be an element of the cpo $Dom_{[\![\Gamma \vdash \sigma:*]\!]\eta}$. Here the environment η is a function which gives the meanings of the free constructor and the free term variables occurring in Γ, so $\eta \in (\mathcal{V}_{cons} \cup \mathcal{V}_{term}) \longrightarrow (\bigcup_\kappa Kind_\kappa \cup \bigcup_a Dom_a)$.
We say that an environment η satisfies a context Γ, again witten $\Gamma \models \eta$, if $\eta(\alpha) \in Kind_\kappa$ for all $\alpha : \kappa$ in Γ and $\eta(x) \in Dom_{[\![\Gamma \vdash \sigma:*]\!]\eta}$ for all $x : \sigma$ in Γ.
For these environments we define the semantics of term expressions, by induction on their type derivation, as follows:

$$\begin{aligned}
[\![\Gamma \vdash x : \sigma]\!]\eta &= \eta(x) && (1)\\
[\![\Gamma \vdash c : \sigma]\!]\eta &= \mathcal{I}_{term}(c) && (2)\\
[\![\Gamma \vdash MN : \tau]\!]\eta &= (\Phi_s [\![\Gamma \vdash M : \sigma \rightarrow \tau]\!]\eta)\ [\![\Gamma \vdash N : \sigma]\!]\eta && (3)\\
[\![\Gamma \vdash (\lambda x : \sigma.M) : \sigma \rightarrow \tau]\!]\eta &= \Phi_s^{-1}(\boldsymbol{\lambda}\xi \in Dom_{[\![\Gamma \vdash \sigma:*]\!]\eta}.[\![\Gamma, x : \sigma \vdash M : \tau]\!]\eta[x := \xi]) && (4)\\
[\![\Gamma \vdash M\sigma : f\sigma]\!]\eta &= (\Phi_t [\![\Gamma, \alpha : * \vdash M : \Pi(\Lambda\alpha : *.\tau)]\!]\eta)\ [\![\Gamma \vdash \sigma : *]\!]\eta && (5)\\
[\![\Gamma \vdash (\Lambda\alpha : *.M) : \Pi(\Lambda\alpha : *.\tau)]\!]\eta &= \Phi_t^{-1}(\boldsymbol{\lambda}a \in Kind_*.[\![\Gamma, \alpha : * \vdash M : \tau]\!]\eta[\alpha := a]) && (6)\\
[\![\Gamma \vdash M : \sigma]\!]\eta &= [\![\Gamma \vdash M : \tau]\!]\eta \quad \text{if} \quad \Gamma \vdash \sigma =_c \tau && (7)
\end{aligned}$$

Here s is $[\![\Gamma \vdash \sigma \rightarrow \tau : *]\!]\eta$ and t is $[\![\Gamma \vdash \Pi(\Lambda\alpha : *.\tau) : *]\!]\eta$.
We require that the ranges of the Φ_a are large enough, so that the right-hand sides of (4) and (6), which involve a Φ^{-1}, are always defined.

There may be several derivations for $\Gamma \vdash M : \sigma$, but becuse terms have a unique type it can easily be proved that all type derivations give the same meaning. For this general model definition we can prove type soundness,

$$[\![\Gamma \vdash M : \sigma]\!] \eta \in Dom_{[\![\Gamma \vdash \sigma : *]\!] \eta}$$

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

In this section we will construct a cpo model for $\boldsymbol{A}$.

First we consider the submodel for the constructor expressions. For this we can use a simple term model. So $[\![\Gamma \vdash \sigma : \kappa]\!] \eta$ is the equivalence class of constructor expressions $\beta\eta$-equal to the expression obtained by substituting $\eta(\alpha)$ for α in σ, for all free constructor variables α.

notation To keep things readable, we write $\sigma \to \tau$ for $[\![\Gamma \vdash \sigma \to \tau : *]\!] \eta$, Πf for $[\![\Gamma \vdash \Pi f : *]\!] \eta$ and fa for $[\![\Gamma, \alpha : * \vdash f\alpha : *]\!] \eta[\alpha := a]$. These abbreviations will be used throughout this paper, whenever we are dealing with a term model as the submodel for the constructor expressions. When we have a different constuctor model, or when we are discussing a general model definition, we will write $[\![\sigma \to \tau]\!]$, $[\![\Pi f]\!]$ and $[\![f\alpha]\!]$.

Because of the general model definition we have given, there only remains the task of finding a family of cpos $Dom = < Dom_a \mid a \in Kind_* >$, that solves the system of coupled domain equations:

$$\forall_{\, a \in Kind_*} : \qquad Dom_a \;\cong\; F_a(Dom) \tag{i}$$

with the associated continuous bijections $\Phi_a \in Dom_a \to F_a(Dom)$.

We use a standard technique, described in [SP82], to find a solution for the recursive domain equations. For this some category theory is needed. A clear and self-contained presentation of this technique can be found in [BH88].

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 $F : \mathcal{K} \to \mathcal{K}$ is a pair (D, ϕ), where D is a $\mathcal{K}$-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.

This result can be used to find a solution of a recursive domain equation. Because of the interdependence of the domain equations, we have to solve all of them together. We consider one recursive domain equation $D \cong F(D)$, where D is a family of cpos. We will construct a solution in a *product category*.

product categories

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

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

If C is an ω-category, then so is $\prod_{a\in I} C$ (see [HS73], [tEH89b]).

For every $b \in I$ we have a projection functor P_b from $\mathcal{K}$ to C, which selects the b-component of a $\mathcal{K}$-object or morphism, i.e. $P_b(< X_a \mid a \in Kind_* >) = X_b$. The projection functors are ω-continuous (see [tEH89b]).

A functor F from $\mathcal{K}$ to $\mathcal{K}$ can be considered as a family of functors $< F_a \mid a \in I >$, where every F_a is a functor from $\mathcal{K}$ to C. It is easily shown that F is ω-continuous iff every component F_a is ω-continuous (see [tEH89b]).

Tupling of functors will be denoted by $< \, , \, >$. For example, $< P_a, P_b >: \mathcal{K} \to C \times C$ is the functor which selects the a and b components of a $\mathcal{K}$-object or morphism.

the construction of a second order model

$\underline{CPO}$ 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
  $FS(f, g) \;=\; (\boldsymbol{\lambda}\xi \in [D \to E].g\circ\xi\circ f) \;\in\; [[D \to E] \to [D' \to E']]$

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

- $GP: \prod_{a\in I} \underline{CPO} \to \underline{CPO}$
- if $< D_a \mid a \in I >$ is a family of cpos, then $GP(< D_a \mid a \in Kind_* >) = \prod_{a\in I} D_a$, the cpo which is the product of all the cpos D_a, with the ordering coordinatewise.
- if $< f_a \mid a \in I >$ is a family of functions, where $f_a \in [D_a \to E_a]$ for all $a \in I$, then
  $GP(< f_a \mid a \in I >) \;=\; \boldsymbol{\lambda}(< d_a \mid a \in I >) \in GP(< D_a \mid a \in I >). < f_a(d_a) \mid a \in I >$ which is a continuous function from $GP(< D_a \mid a \in I >)$ to $GP(< E_a \mid a \in I >)$.

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

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 $\underline{CPO}_{PR}$.
$\underline{CPO}_{PR}$ 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 $\phi : A \to B$ and $\psi : B \to A$ such that $\psi \circ \phi = id_A$ and $\phi \circ \psi \sqsubseteq id_B$.
$\underline{CPO}_{PR}$ is an ω-category (see [SP82], [BH88]).

The functors corresponding with FS and GP are $FS_{PR} : \underline{CPO}_{PR} \times \underline{CPO}_{PR} \to \underline{CPO}_{PR}$ and $GP_{PR} : \prod_{a \in I} \underline{CPO}_{PR} \to \underline{CPO}_{PR}$. They are defined as follows

$$\begin{array}{rcl} FS_{PR}(D,E) & = & FS(D,E) \\ FS_{PR}((\phi,\psi),(\phi',\psi')) & = & (FS(\psi,\phi'), FS(\phi,\psi')) \end{array}$$

and

$$\begin{array}{rcl} GP_{PR}(< D_a \mid a \in I >) & = & GP(< D_a \mid a \in I >) \\ GP_{PR}(< (\phi_a,\psi_a) \mid a \in I >) & = & (GP(< \phi_a \mid a \in I >), GP(< \psi_a \mid a \in I >)) \end{array}$$

So the object parts are unchanged. FS_{PR} and GP_{PR} are ω-continuous (see [SP82] or [BH88] for FS_{PR}, and [tEH89b] for GP_{PR}).

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

We will construct Dom in the product category $\mathcal{K} = \prod_{a \in Kind_*} \underline{CPO}_{PR}$.
We define $F : \mathcal{K} \to \mathcal{K}$ by

$$F = < F_a \mid a \in Kind_* >$$

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

$$\begin{array}{rcl} F_\sigma & = & C_{domain_\sigma} \\ F_{\sigma \to \tau} & = & FS_{PR} \circ < P_\sigma, P_\tau > \\ F_{\Pi f} & = & GP_{PR} \circ < P_{fa} \mid a \in Kind_* > \end{array}$$

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 $\prod \underline{CPO}_{PR}$. 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 $\underline{CPO}_{PR}$ (i.e. $\Psi_a = \Phi_a^{-1}$) . So Dom solves the recursive domain equations, and the embeddings $\Phi_a : Dom_a \to F_a(Dom)$ are the bijections we need.
So an initial fixed point of F gives a family of cpos Dom that satisfies the recursive domain equations and the associated bijections.

So, recapitulating,

- $\underline{CPO}_{PR}$ is an ω-category
- $\prod_{a \in Kind_*} \underline{CPO}_{PR}$ is an ω-category
- FS_{PR}, GP_{PR}, C_A and P_a are ω-continuous
- for all $a \in Kind_*$ the functor $F_a : \prod \underline{CPO}_{PR} \to \underline{CPO}_{PR}$ is ω-continuous

- the functor $F =< F_a \mid a \in Kind_* >: \prod \underline{CPO}_{PR} \rightarrow \prod \underline{CPO}_{PR}$ is ω-continuous
- in $\prod_{a \in Kind_*} \underline{CPO}_{PR}$ the equation $D \cong F(D)$ has an initial solution (Dom, m) where $Dom =< Dom_a \mid a \in Kind_* >$ and $m =< m_a \mid a \in Kind_* >$
- $m_a = (\Phi_a, \Psi_a)$ is an isomorphism between Dom_a and $F_a(Dom)$ for all $a \in Kind_*$.

3 Recursive types

In this section we consider the extension of second order lambda calculus with recursive types. As far as the constructors expressions are concerned, we just add another constructor constant μ, of kind $(* \Rightarrow *) \Rightarrow *$. So if $\Gamma \vdash f : * \Rightarrow *$, then $\Gamma \vdash \mu f : *$.
The whole idea behind a recursive type μf is that it is a solution of

$$\mu f \;\approx\; f(\mu f)$$

so that the types μf, $f(\mu f)$, $f(f(\mu f))$, $\ldots$ are equivalent.

For example, if we have an expression M of type μf, where $f \equiv (\Lambda\alpha : *.\alpha \rightarrow int)$. Because of the equivalence between μf and $f(\mu f) = \mu f \;\rightarrow int$, we want to be able to apply M to itself, and the result should be of type int.

This means that for all $\Gamma \vdash f : * \Rightarrow *$ we require that

$$Dom_{[\![\Gamma \vdash \mu f : *]\!]\eta} \;\cong\; Dom_{[\![\Gamma \vdash f(\mu f) : *]\!]\eta} \qquad (i)$$

We will consider three ways to treat recursive types:

$\boldsymbol{\Lambda\mu_1}$ A recursive type μf and its unfolding $f(\mu f)$ are not identified.

Because we want terms to have a unique type, this means that we cannot both have $\Gamma \vdash M : \mu f$ and $\Gamma \vdash M : f(\mu f)$. We introduce explicit coercion operators $fold_{\mu f}$ and $unfold_{\mu f}$ in the syntax of terms. If $\Gamma \vdash M : \mu f$ then $\Gamma \vdash \; unfold_{\mu f} M : f(\mu f)$, and if $\Gamma \vdash M : f(\mu f)$ then $\Gamma \vdash \; fold_{\mu f} M : \mu f$. The meaning of the fold and unfold operators is given by the isomorphism between $Dom_{[\![\Gamma \vdash \mu f : *]\!]\eta}$ and $Dom_{[\![\Gamma \vdash f(\mu f) : *]\!]\eta}$.

$\boldsymbol{\Lambda\mu_2}$ A recursive type μf and its unfolding $f(\mu f)$ are identified.

So $[\![\, \Gamma \vdash \mu f : * \,]\!]\eta = [\![\, \Gamma \vdash f(\mu f) : * \,]\!]\eta = [\![\, \Gamma \vdash f(f(\mu f)) : * \,]\!]\eta = \ldots$, which means (i) is trivially satisfied, and if $\Gamma \vdash M : \mu f$ then also $\Gamma \vdash M : f(\mu f)$ and vice versa.

$\boldsymbol{\Lambda\mu_3}$ We interpret recursive types as infinite types. This means we not only identify recursive types and their unfoldings, but that we identify all recursive types that have the same infinite unfolding.

For example, the types $\mu(\Lambda\alpha : *.\alpha \rightarrow int)$ and $\mu(\Lambda\alpha : *.(\alpha \rightarrow int) \rightarrow int)$, will be identified, because if we keep unfolding them they both have the same "limit", namely
$(((((\ldots) \rightarrow int) \rightarrow int) \rightarrow int) \rightarrow int) \rightarrow int$.

In $\boldsymbol{\Lambda\mu_2}$ these types would not be identified , because by unfolding them we can never get the same term: unfoldings of the first type will be of the form
$((\ldots(\mu(\Lambda\alpha : *.\alpha \rightarrow int) \ldots \rightarrow int) \rightarrow int$
and unfoldings of the second type will be of the form
$((\ldots(\mu(\Lambda\alpha : *.(\alpha \rightarrow int) \rightarrow int) \ldots \rightarrow int) \rightarrow int$

In the next three sections we will consider how for each of these systems the general model definition and the construction of the cpo model given in part 2 are affected. The general model definition will be changed for each system, and we will alter the construction of the cpo model accordingly.

3.1 $\Lambda\mu_1$

3.1.1 Syntax

constructors

The definition of constructor expressions is unchanged. We just have a new constructor constant, $\mu : (* \Rightarrow *) \Rightarrow *$, for making recursive types.
Constructor equality is $\beta\eta$-equality.

terms

The set of pseudo-terms over $\mathcal{C}_{term}$ and $\mathcal{V}_{term}$ is now defined by

$$M = c \mid x \mid (\lambda x : \sigma.M) \mid M_1 M_2 \mid (\Lambda\alpha : *.\sigma) \mid M\sigma \mid \ fold_{\mu f} M \mid \ unfold_{\mu f} M$$

where $c \in \ \mathcal{C}_{term}$, $x \in \ \mathcal{V}_{term}$, and σ and f are pseudo-constructors.
We have two additional type inference rules

$$\frac{\Gamma \vdash M : \mu f}{\Gamma \vdash \ unfold_{\mu f} M : f(\mu f)}\ (UNFOLD) \quad \text{and} \quad \frac{\Gamma \vdash M : f(\mu f)}{\Gamma \vdash \ fold_{\mu f} M : \mu f}\ (FOLD)$$

As remarked in [tEM88], the subscript μf of $fold_{\mu f}$ is necessary. If it is omitted, some terms no longer have a unique type. This is shown in the following example.

> Suppose $\Gamma \vdash M : f(\mu f)$. Using $(FOLD)$ we can derive two types for $fold\ M$: the type μf of course, but also the type μg, $g \equiv (\Lambda\alpha : *.f(\mu f))$, where α is a fresh type variable. Since α does not occur in $f(\mu f)$, $f(\mu f) = f(\mu f)\ [\alpha := \mu g] \ = \ g(\mu g)$.

We needed the fact that terms have a unique type to guarantee that the definition of the meaning of a term was unambiguous. Therefore $fold$ is written with the subscript μf. By symmetry, we also write $unfold$ with a subscript μf. This subscript, however, could be omitted without causing any problems.

We redefine term equality for $\Lambda\mu_1$. Term equality is the congruence relation generated by $\beta\eta$-equality and the following two rules:

$$\begin{aligned} fold_{\mu f}(\ unfold_{\mu f} M) &= M \\ unfold_{\mu f}(\ fold_{\mu f} M) &= M \end{aligned}$$

3.1.2 Semantics : general model definition

The definition of the semantics of constructor expressions can remain unchanged, since we have no new rules for kind derivations.
We do have new rules for the type inference system. We have to define the meaning of terms that are typed using the new type inference rules $(FOLD)$ and $(UNFOLD)$.
For this we require

$$Dom_{[\![\Gamma \vdash \mu f : *]\!]\eta} \ \cong \ Dom_{[\![\Gamma \vdash f(\mu f) : *]\!]\eta}$$

The associated isomorphism, the bijection

$$\Phi_{[\![\Gamma\vdash\mu f:*]\!]\eta} \in Dom_{[\![\Gamma\vdash\mu f:*]\!]\eta} \longrightarrow Dom_{[\![\Gamma\vdash f(\mu f):*]\!]\eta}$$

gives the semantics of folding and unfolding:

$$\begin{array}{rcl}[\![\,\Gamma\vdash\ unfold_{\mu f}M : f(\mu f)\,]\!]\,\eta & = & \Phi_{[\![\Gamma\vdash\mu f:*]\!]\eta}([\![\,\Gamma\vdash M:\mu f\,]\!]\,\eta)\\ [\![\,\Gamma\vdash\ fold_{\mu f}M : \mu f\,]\!]\,\eta & = & \Phi^{-1}_{[\![\Gamma\vdash\mu f:*]\!]\eta}([\![\,\Gamma\vdash M: f(\mu f)\,]\!]\,\eta)\end{array}$$

We extend the definition of $< F_a \mid a \in Kind_* >$ with

$$F_{[\![\Gamma\vdash\mu f:*]\!]\eta}(< D_a \mid a\in Kind_* >) = D_{[\![\Gamma\vdash f(\mu f):*]\!]\eta}$$

5 definition (general model definition $\Lambda\mu_1$)
An environment model for $\boldsymbol{\Lambda\mu_1}$ is defined as for $\boldsymbol{\Lambda}$ (definition 4), except with the definition of $[\![\ \ ]\!]$ extended as above, and with $F =< F_a \mid a \in Kind_* >$ defined by

$$\begin{array}{rcll}F_{[\![\sigma]\!]}(< D_a \mid a\in Kind_* >) & = & domain_\sigma & \text{for base types } \sigma\\ F_{[\![\sigma\to\tau]\!]}(< D_a \mid a\in Kind_* >) & = & [D_{[\![\sigma]\!]} \longrightarrow D_{[\![\tau]\!]}] & \\ F_{[\![\Pi f]\!]}(< D_a \mid a\in Kind_* >) & = & \prod_{[\![\alpha]\!]\in Kind_*} D_{[\![f\alpha]\!]} & \\ F_{[\![\mu f]\!]}(< D_a \mid a\in Kind_* >) & = & D_{[\![f(\mu f)]\!]} & \end{array}$$

□

3.1.3 The construction of a cpo model

The definition for the submodel for the constructors is the same as it was for $\boldsymbol{\Lambda}$, so we can use the same submodel we used for $\boldsymbol{\Lambda}$, i.e. a term model.

To complete the model we have to construct a family of cpos $Dom =< Dom_a \mid a \in Kind_* >$, that solves the system of coupled domain equations

$$\forall_{\,a\in Kind_*}:\quad Dom_a \cong F_a(Dom)$$

We have the additional domain equations

$$Dom_{[\![\Gamma\vdash\mu f:*]\!]\eta} \cong Dom_{[\![\Gamma\vdash f(\mu f):*]\!]\eta}$$

for all $\Gamma\vdash f : * \Rightarrow *$.

Now we have seen how we found a solution of the system of coupled domain equations for $\boldsymbol{\Lambda}$, solving the new system of coupled domain equations for $\boldsymbol{\Lambda\mu_1}$ is completely straightforward.

We define a new functor $F : \mathcal{K} \to \mathcal{K}$ by $F =< F_a \mid a \in Kind_* >$, where the $F_a : \mathcal{K} \to \underline{CPO}_{PR}$ are defined as follows

$$\begin{array}{rcll}F_\sigma & = & C_{domain_\sigma} & \text{for base types } \sigma\\ F_{\sigma\to\tau} & = & FS_{PR}\circ < P_\sigma, P_\tau > & \\ F_{\Pi f} & = & GP_{PR}\circ < P_{fa} \mid a \in Kind_* > & \\ F_{\mu f} & = & P_{f(\mu f)} & \end{array}$$

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

3.2 $\Lambda\mu_2$

3.2.1 Syntax

We define constructor equality, $=_c$, as the equivalence relation induced by $\beta\eta$-equality and by

$$\Gamma \vdash \ \mu f =_\mu f(\mu f) \qquad \text{for all } \Gamma \vdash f : * \Rightarrow *$$

We call the equality induced by the rule above μ-equality.

Using the type conversion rule

$$\frac{\Gamma \vdash M : \sigma \quad \Gamma \vdash \sigma =_c \tau}{\Gamma \vdash M : \tau} \ (TEQ)$$

we can derive

$$\frac{\Gamma \vdash M : \mu f}{\Gamma \vdash M : f(\mu f)} \qquad \frac{\Gamma \vdash M : f(\mu f)}{\Gamma \vdash M : \mu f}$$

So we can drop the $fold_{\mu f}$ and $unfold_{\mu f}$ from our syntax, and we no longer need the extra domain equations

$$Dom_{[\![\Gamma \vdash \mu f : *]\!]\eta} \cong Dom_{[\![\Gamma \vdash f(\mu f) : *]\!]\eta}$$

we needed for $\Lambda\mu_1$, since $\mu f =_c f(\mu f)$, and so $[\![\Gamma \vdash \mu f : *]\!]\eta = [\![\Gamma \vdash f(\mu f) : *]\!]\eta$.

3.2.2 Semantics : general model definition

We can take the same recursive domain equations we had for Λ:

$$\forall_{\ a \in Kind_*} : \quad Dom_a \cong F_a(Dom)$$

where

$$\begin{array}{rcll} F_{[\![\sigma]\!]}(< D_a \mid a \in Kind_* >) & = & domain_\sigma & \text{for base types } \sigma \\ F_{[\![\sigma \to \tau]\!]}(< D_a \mid a \in Kind_* >) & = & [D_{[\![\sigma]\!]} \longrightarrow D_{[\![\tau]\!]}] & \\ F_{[\![\Pi f]\!]}(< D_a \mid a \in Kind_* >) & = & \prod_{[\![\alpha]\!] \in Kind_*} D_{[\![f\alpha]\!]} & \end{array}$$

For the recursive types these domain equations achieve precisely what we want them to. Because $\mu f =_\mu f(\mu f)$, the constructor model should satisfy $[\![\Gamma \vdash \mu f : *]\!]\eta = [\![\Gamma \vdash f(\mu f) : *]\!]\eta$ and for the recursive type $\mu f \equiv \mu(\Lambda\alpha : *.\alpha \to int)$ we then get

$$\begin{array}{rcll} Dom_{[\![\mu f]\!]} & = & Dom_{[\![\mu f \to int]\!]} & \text{, since } [\![\mu f]\!] = [\![f(\mu f)]\!] = [\![\mu f \to int]\!] \\ & \cong & [Dom_{[\![\mu f]\!]} \to Dom_{[\![int]\!]}] & \\ & = & [Dom_{[\![\mu f \to int]\!]} \to Dom_{[\![int]\!]}] & \\ & \cong & \cdots & \end{array}$$

If $\mu f =_{\beta\eta\mu} \mu(\Lambda\alpha : *.\alpha)$ then $F_{[\![\mu f]\!]}$ is as yet undefined. We take

$$F_{[\![\mu(\Lambda\alpha:*.\alpha)]\!]}(< D_a \mid a \in Kind_* >) = D_{[\![\mu(\Lambda\alpha:*.\alpha)]\!]}$$

This means the domain equation for $\mu(\Lambda\alpha : *.\alpha)$ is

$$Dom_{[\![\mu(\Lambda\alpha:*.\alpha)]\!]} \cong F_{[\![\mu(\Lambda\alpha:*.\alpha)]\!]}(< Dom_{[\![\Gamma \vdash a:*]\!]\eta} \mid a \in Kind_* >) = Dom_{[\![\mu(\Lambda\alpha:*.\alpha)]\!]}$$

6 definition (general model definition $\Lambda\mu_2$)
An environment model for $\boldsymbol{\Lambda\mu_2}$is defined as for $\boldsymbol{\Lambda}$ (definition 4), except with
$F =< F_a \mid a \in Kind_* >$ defined by

$$\begin{array}{rcll}
F_{[\![\sigma]\!]}(< D_a \mid a \in Kind_* >) & = & domain_\sigma & \text{for base types } \sigma \\
F_{[\![\sigma \to \tau]\!]}(< D_a \mid a \in Kind_* >) & = & [D_{[\![\sigma]\!]} \longrightarrow D_{[\![\tau]\!]}] & \\
F_{[\![\Pi f]\!]}(< D_a \mid a \in Kind_* >) & = & \prod_{[\![\alpha]\!] \in Kind_*} D_{[\![f\alpha]\!]} & \\
F_{[\![\mu(\Lambda\alpha:*.\alpha)]\!]}(< D_a \mid a \in Kind_* >) & = & D_{[\![\mu(\Lambda\alpha:*.\alpha)]\!]} &
\end{array}$$

□

3.2.3 The construction of a cpo model

For the constructors we again take a term model, only this time $[\![\Gamma \vdash \sigma : \kappa]\!]\eta$ is the equivalence class of constructor expressions $\beta\eta\mu$-equal (and not just $\beta\eta$-equal) to σ with all free constructor variables α replaced by $\eta(\alpha)$.
The family of cpos Dom satisfying

$$\forall\, a \in Kind_* : \qquad Dom_a \;\cong\; F_a(Dom)$$

is constructed in the by now familiar way, as the initial fixed point of a functor $F : \mathcal{K} \to \mathcal{K}$, $F =< F_a \mid a \in Kind_* >$, where the $F_a : \mathcal{K} \to \underline{CPO}_{PR}$ are defined by

$$\begin{array}{lcll}
F_\sigma & = & C_{domain_\sigma} & \text{if } \sigma \text{ is a base type} \\
F_{\sigma \to \tau} & = & FS_{PR} \circ < P_\sigma, P_\tau > & \\
F_{\Pi f} & = & GP_{PR} \circ < P_{fa} \mid a \in Kind_* > & \\
F_{\mu(\Lambda\alpha:*.\alpha)} & = & P_{\mu(\Lambda\alpha:*.\alpha)} &
\end{array}$$

The initial fixed point of F gives the cpos Dom_a satisfying the recursive domain equations, and the associated isomorphisms $\Phi_a \in Dom_a \to F_a(Dom)$. The cpo $Dom_{\mu(\Lambda\alpha:*.\alpha)}$ will be the one-point cpo.

3.3 $\Lambda\mu_3$

3.3.1 Syntax

We shall now interpret recursive types as infinite types.
To define the resulting congruence relation on types, we define a tree $\mathcal{T}(\sigma)$ for every type σ. These trees will be *regular* trees, i.e. trees with a finite set of subtrees. The leaves will be base types or types only consisting of constructor variables, and the nodes correspond to type constructors.

7 definition ($\mathcal{T}$)
The function $\mathcal{T}$ from types to regular trees is defined by

$$
\begin{array}{rcl}
\mathcal{T}(\sigma) & = & .\sigma \quad \text{if } \sigma \text{ consists only of constructor variables} \\
\mathcal{T}(c) & = & .c \\
\mathcal{T}(\sigma \to \tau) & = & \begin{array}{ccc} & \longrightarrow & \\ \swarrow & & \searrow \\ \mathcal{T}(\sigma) & & \mathcal{T}(\tau) \end{array} \\
\mathcal{T}(\Pi(\Lambda\alpha : *.\sigma)) & = & \begin{array}{c} \Pi_\alpha \\ \downarrow \\ \mathcal{T}(\sigma) \end{array} \\
\mathcal{T}(\mu(\Lambda\alpha : *.\sigma)) & = & \left\{ \begin{array}{ll} \mathcal{T}(\sigma)\,[\alpha := \mathcal{T}(\mu(\Lambda\alpha : *.\sigma))] & \text{if } \mathcal{T}(\sigma) \neq \alpha \\ \bot & \text{if } \mathcal{T}(\sigma) = \alpha \end{array} \right. \\
\mathcal{T}(\sigma) & = & \mathcal{T}(\tau) \quad \text{if } \sigma =_{\beta\eta} \tau
\end{array}
$$

Note that we have bound type variables in the trees: every Π-node introduces a bound type variable. α-equal trees are identified. $\mathcal{T}(\sigma)[\alpha := \ldots]$ is tree substitution.
□

By the following property it is clear that this defines a regular tree for every type.

8 property ([Cou83] , theorem 4.2.1)
If $t \neq .\alpha$, and t is regular, then there is a unique tree x such that $x = t[\alpha := x]$, and x will be regular. □

Some examples. Suppose $\Gamma \equiv g : * \Rightarrow *\ ,\ \beta : *$. Then

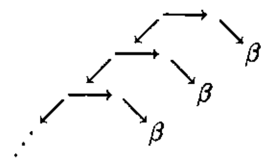
$$
\begin{array}{rcl}
\mathcal{T}(\beta) & = & .\beta \\
\mathcal{T}(g\beta) & = & .g\beta \\
\mathcal{T}(\Pi(\Lambda\alpha : *.\alpha \to \beta)) & = & \begin{array}{ccc} & \Pi_\alpha & \\ & \downarrow & \\ & \longrightarrow & \\ \swarrow & & \searrow \\ \alpha & & \beta \end{array} \\
\mathcal{T}(\mu(\Lambda\alpha : *.\alpha)) & = & \bot
\end{array}
$$

Let $f \equiv (\Lambda\alpha : *.\alpha \to \beta)$. Then

$$
\begin{array}{rcl}
\mathcal{T}(\mu f) & = & \begin{array}{ccc} & \longrightarrow & \\ \swarrow & & \searrow \\ \alpha & & \beta \end{array} \quad [\alpha := \mathcal{T}(\mu f)] \\
& = & \begin{array}{ccc} & \longrightarrow & \\ \swarrow & & \searrow \\ \mathcal{T}(\mu f) & & \beta \end{array}
\end{array}
$$

So

$$\mathcal{T}(\mu f) \ =$$



We would get the same tree for $\mu(\Lambda\alpha : *.(\alpha \rightarrow \beta) \rightarrow \beta)$, $\mu(\Lambda\alpha : *.((\alpha \rightarrow \beta) \rightarrow \beta) \rightarrow \beta))$, etc. .

9 definition $(\approx)$
The equivalence relation $\approx$ on types is defined by
$\sigma \ \approx \ \tau \ \iff \ \mathcal{T}(\sigma) = \mathcal{T}(\tau)$
□

For all types σ and τ, $\sigma \ \approx \ \tau$ is decidable. (This is because $\mathcal{T}(\sigma)$ and $\mathcal{T}(\tau)$ are regular.)
We take $\approx$ as our notion of type equality. Equality for constructor expressions of higher kinds is the congruence relation generated by $\approx$ and the $\beta\eta$ rules. So the type conversion rule *(TEQ)* has become

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

3.3.2 Semantics : general model definition

Again we take the same recursive domain equations as for $\boldsymbol{\Lambda}$, and for the type $\mu(\Lambda\alpha : *.\alpha)$ we add

$$Dom_{[\![\mu(\Lambda\alpha:*.\alpha)]\!]} \ \cong \ Dom_{[\![\mu(\Lambda\alpha:*.\alpha)]\!]}$$

as we did for $\boldsymbol{\Lambda\mu_2}$.

10 definition (general model definition $\Lambda\mu_3$)
An environment model for $\boldsymbol{\Lambda\mu_3}$is defined as for $\boldsymbol{\Lambda}$ (definition 4), except with
$F =< F_a \mid a \in Kind_* >$ defined by

$$\begin{array}{rcll} F_{[\![\sigma]\!]}(< D_a \mid a \in Kind_* >) & = & domain_\sigma & \text{for base types } \sigma \\ F_{[\![\sigma\rightarrow\tau]\!]}(< D_a \mid a \in Kind_* >) & = & [D_{[\![\sigma]\!]} \longrightarrow D_{[\![\tau]\!]}] & \\ F_{[\![\Pi f]\!]}(< D_a \mid a \in Kind_* >) & = & \prod_{[\![\alpha]\!] \in Kind_*} D_{[\![f\alpha]\!]} & \\ F_{[\![\mu(\Lambda\alpha:*.\alpha)]\!]}(< D_a \mid a \in Kind_* >) & = & D_{[\![\mu(\Lambda\alpha:*.\alpha)]\!]} & \end{array}$$

□

3.3.3 The construction of a cpo model

the submodel for the constructor expressions

For $\boldsymbol{\Lambda\mu_3}$we choose a different constructor model: types will be interpreted as trees. The leaves will be base types or type variables, and the nodes correspond to type constructors.
If all free constructor variables in σ are *type* variables, then the meaning of a type σ in environment η will be the tree

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

i.e. $\mathcal{T}(\sigma)$ with all type variables α replaced by $\eta(\alpha)$.

For example

$$
\begin{aligned}
[\![\Gamma \vdash int : *]\!] \eta &= .int \\
[\![\Gamma, \alpha : * \vdash \alpha \rightarrow int : *]\!] \eta &= \begin{tikzcd}[column sep=small] & \longrightarrow \arrow[dl] \arrow[dr] & \\ \eta(\alpha) & & int \end{tikzcd}
\end{aligned}
$$

Let $f \equiv (\Lambda\alpha : *.\alpha \rightarrow \beta)$. Then

$$
[\![\Gamma \vdash \mu f : *]\!] \eta = \begin{tikzcd}[column sep=small] & & & \longrightarrow \arrow[dl] \arrow[dr] & \\ & & \longrightarrow \arrow[dl] \arrow[dr] & & \eta(\beta) \\ & \longrightarrow \arrow[dl] \arrow[dr] & & \eta(\beta) & \\ \cdot^{\cdot^{\cdot}} & & \eta(\beta) & & \end{tikzcd}
$$

$Kind_{*\Rightarrow *}$ will be a subset of $Kind_* \rightarrow Kind_*$, and $[\![\Gamma \vdash \mu f : *]\!] \eta \in Kind_*$ will be a fixed point of $[\![\Gamma \vdash f : * \Rightarrow *]\!] \eta \in Kind_{*\Rightarrow *}$.
For instance, the meaning of f will be :

$$
\begin{aligned}
[\![\Gamma \vdash f : * \Rightarrow *]\!] \eta &= \lambda\!\!\!\lambda a \in Kind_*. [\![\Gamma \vdash \alpha \rightarrow \beta : *]\!] \eta[\alpha := a] \\
&= \lambda\!\!\!\lambda a \in Kind_*. \begin{tikzcd}[column sep=small] & \longrightarrow \arrow[dl] \arrow[dr] & \\ {[\![\Gamma \vdash \alpha : *]\!] \eta[\alpha := a]} & & {[\![\Gamma \vdash \beta : *]\!] \eta} \end{tikzcd} \\
&= \lambda\!\!\!\lambda a \in Kind_*. \begin{tikzcd}[column sep=small] & \longrightarrow \arrow[dl] \arrow[dr] & \\ a & & \eta(\beta) \end{tikzcd}
\end{aligned}
$$

So $[\![\Gamma \vdash \mu f : *]\!] \eta \;=\; [\![\Gamma \vdash f : * \Rightarrow *]\!] \eta \; [\![\Gamma \vdash \mu f : *]\!] \eta$.

We will now define the submodel for the constructors in the way prescribed by the general model definition. So we have to define $Kind = \langle Kind_\kappa \mid \kappa \text{ a kind} \rangle$, and we have to define $\mathcal{I}_{cons}$, giving the meaning of the constructor constants, with $\mathcal{I}_{cons}(c^\kappa) \in Kind_\kappa$.

11 definition (**Tree**)
Tree is the set of all finite and infinite trees with base types and type variables as leaves, and $\rightarrow$ and Π_α, α a type variable, as nodes. $\rightarrow$-nodes have two subtrees, Π_α-nodes have one subtree.
□

We will define a partial order $\sqsubseteq$ on **Tree**, so that for all $\Gamma \vdash f : * \Rightarrow *$ we can define $[\![\Gamma \vdash \mu f : *]\!] \eta$ as the *least* fixed point of $[\![\Gamma \vdash f : * \Rightarrow *]\!] \eta$.

12 definition ($\sqsubseteq$)
$\sqsubseteq$ is a partial order on **Tree** defined by

$$
\begin{array}{ccll}
\perp & \sqsubseteq & a & \text{for all } a \in \mathbf{Tree} \\[2ex]
a & \sqsubseteq & a & \text{for all } a \in \mathbf{Tree} \\[2ex]
\begin{tikzcd}[column sep=small] & \longrightarrow \arrow[dl] \arrow[dr] & \\ a & & b \end{tikzcd} & \sqsubseteq & \begin{tikzcd}[column sep=small] & \longrightarrow \arrow[dl] \arrow[dr] & \\ a' & & b' \end{tikzcd} & \text{if } a \sqsubseteq a' \text{ and } b \sqsubseteq b' \\[2ex]
\begin{tikzcd} \Pi_\alpha \arrow[d] \\ a \end{tikzcd} & \sqsubseteq & \begin{tikzcd} \Pi_\alpha \arrow[d] \\ b \end{tikzcd} & \text{if } a \sqsubseteq b
\end{array}
$$

□

So $a \sqsubseteq b$, if we can get a by cutting of some subtrees of b and replacing them by $\perp$. Every ascending chain in **Tree** has a least upper bound.

$< Kind_\kappa \mid \kappa$ a kind $>$ is defined by

$$\begin{array}{lcl} Kind_* & = & \{\, t \mid t \in \mathbf{Tree} \ \wedge \ FV(t) = \emptyset \} \\ Kind_{*\Rightarrow *} & = & [Kind_* \to Kind_*] \\ Kind_{\kappa_1 \Rightarrow \kappa_2} & = & Kind_{\kappa_1} \to Kind_{\kappa_2} \quad , \kappa_1 \Rightarrow \kappa_2 \neq * \Rightarrow * \end{array}$$

Here $FV(t)$ denotes the set of free type variables occurring in t.

$Kind_{*\Rightarrow *}$ is restricted to functions from $Kind_*$ to $Kind_*$ that are continuous with respect to $\sqsubseteq$, because for all $F \in Kind_{*\Rightarrow *}$ we want $[\![\, f : * \Rightarrow * \vdash \mu f : * \,]\!]\,[f := F]$ to be the initial fixed point of F. For $Kind_{\kappa_1 \Rightarrow \kappa_2}$, $\kappa_1 \Rightarrow \kappa_2 \neq * \Rightarrow *$, we have no such requirements.

The meaning of the constructor constants is given by

$$\begin{array}{lcl} \mathcal{I}_{cons}(\sigma) & = & .\sigma \quad \text{for base types } \sigma \\ \mathcal{I}_{cons}(\to) & = & \boldsymbol{\lambda} a \in Kind_*.\boldsymbol{\lambda} b \in Kind_*. \ \begin{array}{ccc} & \longrightarrow & \\ \swarrow & & \searrow \\ a & & b \end{array} \\ \mathcal{I}_{cons}(\Pi) & = & \boldsymbol{\lambda} F \in Kind_{*\Rightarrow *}.\ \begin{array}{c} \Pi_\alpha \\ \downarrow \\ F\alpha \end{array} \\ \mathcal{I}_{cons}(\mu) & = & \boldsymbol{\lambda} F \in Kind_{*\Rightarrow *}.\bigsqcup_{i \in \mathbb{N}} F^i \perp \end{array}$$

It is easy to see that $\mathcal{I}_{cons}(\sigma) \in Kind_*$, $\mathcal{I}_{cons}(\to) \in Kind_{*\Rightarrow(*\Rightarrow *)}$, $\mathcal{I}_{cons}(\Pi) \in Kind_{(*\Rightarrow *)\Rightarrow *}$ and $\mathcal{I}_{cons}(\mu) \in Kind_{(*\Rightarrow *)\Rightarrow *}$.

The sets $Kind_\kappa$ are actually larger than they have to be. The following $Kind'_\kappa$ could also be used

$$\begin{array}{lcl} Kind'_* & = & \{\, t \mid t \in \mathbf{Tree} \ \wedge \ FV(t) = \emptyset \wedge \ t \text{ is regular } \} \\ Kind'_{*\Rightarrow *} & = & \{\, (\boldsymbol{\lambda} a \in Kind'.t) \mid t \in \mathbf{Tree} \ \wedge \ FV(t) \subseteq \{a\} \wedge \ t \text{ is regular } \} \\ Kind'_{\kappa_1 \Rightarrow \kappa_2} & = & Kind'_{\kappa_1} \to Kind'_{\kappa_2} \quad , \kappa_1 \Rightarrow \kappa_2 \neq * \Rightarrow * \end{array}$$

Clearly $\mathcal{I}_{cons}(\sigma) \in Kind'_*$, $\mathcal{I}_{cons}(\to) \in Kind'_{*\Rightarrow(*\Rightarrow *)}$ and $\mathcal{I}_{cons}(\Pi) \in Kind'_{(*\Rightarrow *)\Rightarrow *}$.
That $\mathcal{I}_{cons}(\mu) \in Kind'_{(*\Rightarrow *)\Rightarrow *}$ is an immediate consequence of the following lemma.

13 lemma For $F \in Kind'_{*\Rightarrow *}$, $\bigsqcup_{i \in \mathbb{N}} F^i \perp \ \in Kind'_*$

proof

Suppose $F = (\boldsymbol{\lambda} a \in Kind'_*.t) \ \in Kind'_{*\to *}$. This means $Ft' = t[a := t']$, so
$F\perp = t[a := \perp]$, $F^2\perp = t[a := F\perp]$, $F^3\perp = t[a := F^2\perp]$, ...
and therefore $\perp \sqsubseteq F\perp \sqsubseteq F^2\perp \sqsubseteq \ \ldots$. This chain has a lub, $\bigsqcup_{i \in \mathbb{N}} F^i \perp \in \mathbf{Tree}$.
There remains to be shown that $\bigsqcup F^i \perp \in Kind'_*$, i.e. that $\bigsqcup F^i \perp$ contains no free variables, and that $\bigsqcup F^i \perp$ is regular.
Clearly $\bigsqcup F^i \perp$ does not contain free variables, since all $F^i \perp \in Kind'_*$, so none of them contain free variables.
To prove that $\bigsqcup F^i \perp$ is regular, we distinguish between $t = a$ and $t \neq a$. The former case is trivial. If $t \neq a$, then the equation $x = t[a := x]$ has a unique, regular, solution (property 8). But since F is continuous (see [Cou83], proposition 3.3.3), $\bigsqcup F^i \perp = F(\bigsqcup F^i \perp) = t[a := \bigsqcup F^i \perp]$,
that solution must be $\bigsqcup F^i \perp$.

□

the model for the terms

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

$$\forall_{\ a \in Kind_*}: \qquad Dom_a \ \cong\ F_a(Dom)$$

i.e.

$$\begin{array}{lcll}
Dom_{[\![\sigma]\!]} & = & Dom_{.\sigma} & \cong\ domain_\sigma \\
Dom_{[\![\sigma \to \tau]\!]} & = & Dom_{\substack{\longrightarrow \\ \swarrow\quad\searrow \\ [\![\sigma]\!]\quad\ [\![\tau]\!]}} & \cong\ [Dom_{[\![\sigma]\!]} \longrightarrow Dom_{[\![\tau]\!]}] \\
Dom_{[\![\Pi f]\!]} & = & Dom_{\substack{\Pi_\alpha \\ \downarrow \\ [\![f\alpha]\!]}} & \cong\ \prod_{[\![\alpha]\!] \in Kind_*} Dom_{[\![f\alpha]\!]} \\
Dom_{[\![\mu(\Lambda\alpha:*.\alpha)]\!]} & = & Dom_\perp & \cong\ Dom_\perp
\end{array}$$

We define the functor $F: \mathcal{K} \to \mathcal{K}$ by $F = < F_a \mid a \in Kind_* >$, where the functors $F_a : \mathcal{K} \to \underline{CPO}_{PR}$ are defined by

$$\begin{array}{lcll}
F_{.\sigma} & = & C_{domain_\sigma} & \text{for base types } \sigma \\
F_{\substack{\longrightarrow \\ \swarrow\quad\searrow \\ \sigma\quad\ \tau}} & = & FS_{PR} \circ < P_\sigma, P_\tau > & \\
F_{\substack{\Pi_\alpha \\ \downarrow \\ \tau}} & = & GP_{PR} \circ < P_{\tau[\alpha:=a]} \mid a \in Kind_* > & \\
F_\perp & = & P_\perp &
\end{array}$$

The initial fixed point of F gives the cpos Dom_a satisfying the recursive domain equations, and the associated isomorphisms $\Phi_a \in Dom_a \to F_a(Dom)$. Again, the cpo $Dom_{[\![\mu(\Lambda\alpha:*.\alpha)]\!]}$ will be the one-point cpo.

4 Subtyping

We now consider the extension of system $\boldsymbol{A}$ with subtyping. This system will be called $\boldsymbol{A}_{\leq}$.

4.1 Syntax

We will have a subtype relation $\leq$ on types. If $\sigma \leq \tau$, we say that σ is a subtype of τ. The subtype relation will be a pre-order (ie. reflexive and transitive).
We add the following type inference rule: the ***subsumption rule***

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

This means that terms no longer have a unique type.

The subtype relation will be based on a subtype relation $\leq^B$ on the base types. For example, if *int* and *real* are base types , we could have *int* $\leq^B$ *real*.

We have the following rules for deducing $\sigma \leq \tau$ for more complex types σ and τ.

$$\frac{\sigma \leq^B \tau}{\Gamma \vdash \sigma \leq \tau}(START) \qquad \frac{\Gamma \vdash \sigma =_c \tau}{\Gamma \vdash \sigma \leq \tau}(REFL) \qquad \frac{\Gamma \vdash \rho \leq \sigma \quad \Gamma \vdash \sigma \leq \tau}{\Gamma \vdash \rho \leq \tau}(TRANS)$$

$$\frac{\Gamma \vdash \sigma' \leq \sigma \quad \Gamma \vdash \tau \leq \tau'}{\Gamma \vdash \sigma \rightarrow \tau \leq \sigma' \rightarrow \tau'}(\leq\rightarrow) \qquad \frac{\Gamma, \alpha : * \vdash f\alpha \leq g\alpha}{\Gamma \vdash \Pi f \leq \Pi g}(\leq \Pi)$$

Note the contravariance of $\rightarrow$ with respect to the subtype relation.
That $\leq$ is indeed a pre-order is of course guaranteed by the rule $(REFL)$ and $(TRANS)$. The rule *(TEQ)* is omitted, since it can be derived from $(REFL)$ and *(SUB)*.
For the model construction we will need the following lemma.

14 lemma
$$\begin{array}{rcl} \Gamma \vdash \sigma \rightarrow \tau \ \leq \ \sigma' \rightarrow \tau' & \Longrightarrow & \Gamma \vdash \sigma' \leq \sigma \text{ and } \Gamma \vdash \tau \leq \tau' \\ \Gamma \vdash \Pi f \ \leq \ \Pi g & \Longrightarrow & \Gamma, \alpha : * \vdash f\alpha \leq g\alpha \end{array}$$
□

This lemma can be proved as follows. We define a relation $\leq'$ on types. For $\leq'$ we have the same derivation rules as for $\leq$, except instead of $(TRANS)$ we have the following rule

$$\frac{\Gamma \vdash \sigma \leq' \tau \quad \Gamma \vdash \sigma =_c \sigma' \quad \Gamma \vdash \tau =_c \tau'}{\Gamma \vdash \sigma' \leq' \tau'}(\leq TEQ)$$

Clearly $\Gamma \vdash \sigma \leq' \tau \ \Rightarrow \ \Gamma \vdash \sigma \leq \tau$.
By the next lemma we also have $\Gamma \vdash \sigma \leq \tau \ \Rightarrow \ \Gamma \vdash \sigma \leq' \tau$.

15 lemma $\leq'$ is transitive, ie. $\Gamma \vdash \rho \leq' \sigma \ \& \ \Gamma \vdash \sigma \leq' \tau \ \Rightarrow \ \Gamma \vdash \rho \leq' \tau$
proof By induction on the derivation length, not counting the rule $(\leq TEQ)$.
Suppose $\Gamma \vdash \rho \leq' \sigma$ and $\Gamma \vdash \sigma \leq' \tau$. Then

(a) ρ,σ and τ are $\beta\eta$-equal to base types α,β and γ, respectively, and $\alpha \leq^B \beta \leq^B \gamma$, or

(b) $\rho =_c \rho_1 \rightarrow \rho_2$, $\sigma =_c \sigma_1 \rightarrow \sigma_2$ and $\tau =_c \tau_1 \rightarrow \tau_2$, or

(c) $\rho =_c \Pi f$, $\sigma =_c \Pi g$ and $\tau =_c \Pi h$.

We must prove $\Gamma \vdash \rho \leq' \tau$. For (a) this is trivial. We will give the proof for (b). The proof for (c) is similar.

The derivations of $\Gamma \vdash \rho \leq' \sigma$ and $\Gamma \vdash \sigma \leq' \tau$, must both end with $(\leq \rightarrow)$, possibly followed by $(\leq TEQ)$. So $\Gamma \vdash \sigma_1 \leq' \rho_1$, $\Gamma \vdash \rho_2 \leq' \sigma_2$, $\Gamma \vdash \tau_1 \leq' \sigma_1$ and $\Gamma \vdash \sigma_2 \leq' \tau_2$.
By the induction hypothesis $\Gamma \vdash \tau_1 \leq' \rho_1$ and $\Gamma \vdash \rho_2 \leq' \tau_2$, and hence $\Gamma \vdash \rho \leq' \tau$
□

So $\Gamma \vdash \sigma \leq' \tau \Leftrightarrow \Gamma \vdash \sigma \leq \tau$, and for $\leq'$ it is obvious that lemma 14 holds. In fact, we already used it in the proof of lemma 15.

4.2 Semantics : general model definition

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 from $\Gamma \vdash M : \sigma$ and $\Gamma \vdash \sigma \leq \tau$:

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

Since $[\![\Gamma \vdash M : \sigma]\!] \eta \in Dom_{[\![\Gamma \vdash \sigma : *]\!] \eta}$ and we want $[\![\Gamma \vdash M : \tau]\!] \eta \in Dom_{[\![\Gamma \vdash \tau : *]\!] \eta}$, we need a *coercion function* from $Dom_{[\![\Gamma \vdash \sigma : *]\!] \eta}$ to $Dom_{[\![\Gamma \vdash \tau : *]\!] \eta}$. We will call this function $Coe_{[\![\Gamma \vdash \sigma : *]\!] \eta \; [\![\Gamma \vdash \tau : *]\!] \eta}$
.

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

$$[\![\Gamma \vdash M : \tau]\!] \eta \;=\; Coe_{[\![\Gamma \vdash \sigma : *]\!] \eta \; [\![\Gamma \vdash \tau : *]\!] \eta} \; [\![\Gamma \vdash M : \sigma]\!] \eta$$

For all types σ and τ such that $\Gamma \vdash \sigma \leq \tau$, we need a coercion function from $Dom_{[\![\Gamma \vdash \sigma : *]\!] \eta}$ to $Dom_{[\![\Gamma \vdash \tau : *]\!] \eta}$. We require that the coercion functions are continuous.

Not any set of coercion function will do. Remember that the meaning of a term is defined by induction on its type derivation. 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. We have to prove *coherence*, that all derivations for $\Gamma \vdash M : \sigma$ give the same meaning $[\![\Gamma \vdash M : \sigma]\!] \eta$.

We will now try to find some additional requirements for the coercion functions to guarantee that an environment model is coherent.

notation The same same conventions we use to abbreviate the subscripts of the form Dom will be used for the subscripts of Coe and Φ. So we will write $Coe_{[\![\sigma]\!] [\![\tau]\!]}$ instead of $Coe_{[\![\Gamma \vdash \sigma : *]\!] \eta \; [\![\Gamma \vdash \tau : *]\!] \eta}$ and $\Phi_{[\![\sigma]\!]}$ instead of $\Phi_{[\![\Gamma \vdash \sigma : *]\!] \eta}$. When we are dealing with the term model for the constructor expressions, we will just write $Coe_{\sigma \; \tau}$ and Φ_σ.

coherence

The subsumption rule itself gives rise to the following two fairly obvious requirements for the coercion fuctions:

$$\begin{array}{lll} \mathcal{P}_0 & Coe_{a \; a} = \boldsymbol{\lambda} \xi \in Dom_a . \xi & \text{for all } a \in Kind_* \\ \mathcal{P}_1 & Coe_{a \; c} = Coe_{b \; c} \circ Coe_{a \; b} & \text{for all } a \leq^* b \leq^* c \end{array}$$

16 lemma If $\mathcal{P}_0$ or $\mathcal{P}_1$ does not hold, then the semantics is not coherent.
proof The subtype relation is reflexive, so

$$\frac{\Gamma \vdash M : \sigma \quad \Gamma \vdash \sigma \leq \sigma}{\Gamma \vdash M : \sigma}$$

which yields

$$[\![\Gamma \vdash M : \sigma]\!] \eta = Coe_{[\sigma][\sigma]} [\![\Gamma \vdash M : \sigma]\!] \eta \ ,$$

So if $\mathcal{P}_0$ does not hold, than $\Gamma \vdash M : \sigma$ does not have a unique meaning.
Suppose $\Gamma \vdash \rho \leq \sigma \leq \tau$. Then

$$\frac{\Gamma \vdash M : \rho \qquad \Gamma \vdash \rho \leq \tau}{\Gamma \vdash M : \tau}$$

yields $[\![\Gamma \vdash M : \tau]\!] \eta = Coe_{[\rho][\tau]} [\![\Gamma \vdash M : \rho]\!] \eta$
but

$$\frac{\dfrac{\Gamma \vdash M : \rho \quad \Gamma \vdash \rho \leq \sigma}{\Gamma \vdash M : \sigma} \quad \Gamma \vdash \sigma \leq \tau}{\Gamma \vdash M : \tau}$$

yields
$[\![\Gamma \vdash M : \tau]\!] \eta = Coe_{[\sigma][\tau]}(Coe_{[\rho][\sigma]} [\![\Gamma \vdash M : \rho]\!] \eta)$
So if $\mathcal{P}_1$ does not hold, than $\Gamma \vdash M : \tau$ does not have a unique meaning. □

$\mathcal{P}_0$ and $\mathcal{P}_1$ are not sufficient to have coherence. We will also require properties of the coercions between $\rightarrow$-types and Π-types.

First we consider function types. Let $\sigma' \leq \sigma$ and $\tau \leq \tau'$, so $\sigma \rightarrow \tau \leq \sigma' \rightarrow \tau'$. Suppose $\Gamma \vdash M : \sigma \rightarrow \tau$ and $\Gamma \vdash N : \sigma'$. Then $\Gamma \vdash MN : \tau'$ can be derived in several ways, for instance:

$$\text{(i)} \quad \frac{\dfrac{M : \sigma \rightarrow \tau \quad \sigma \rightarrow \tau \leq \sigma' \rightarrow \tau'}{M : \sigma' \rightarrow \tau'} \quad N : \sigma'}{MN : \tau'} \qquad \text{(ii)} \quad \frac{\dfrac{M : \sigma \rightarrow \tau \quad \dfrac{N : \sigma' \quad \sigma' \leq \sigma}{N : \sigma}}{MN : \tau} \quad \tau \leq \tau'}{MN : \tau'}$$

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

$$(\Phi(Coe_{[\sigma \rightarrow \tau][\sigma' \rightarrow \tau']} [\![\Gamma \vdash M : \sigma \rightarrow \tau]\!] \eta)) \ [\![\Gamma \vdash N : \sigma']\!] \eta \qquad \text{(i)}$$

$$Coe_{[\tau][\tau']}((\Phi [\![\Gamma \vdash M : \sigma \rightarrow \tau]\!] \eta) \ (Coe_{[\sigma'][\sigma]} [\![\Gamma \vdash N : \sigma']\!] \eta)) \qquad \text{(ii)}$$

In order for these to be equal, some equation between $Coe_{[\sigma \rightarrow \tau][\sigma' \rightarrow \tau']}$ and $Coe_{[\sigma'][\sigma]}$ and $Coe_{[\tau][\tau']}$ has to hold. There is really only one way to express a relation between $Coe_{[\sigma \rightarrow \tau][\sigma' \rightarrow \tau']}$ and $Coe_{[\sigma'][\sigma]}$ and $Coe_{[\tau][\tau']}$.

$$\begin{array}{lcllll}
Dom_{[\sigma \rightarrow \tau]} & \cong & [& Dom_{[\sigma]} & \longrightarrow & Dom_{[\tau]} &] \\
\downarrow Coe_{[\sigma \rightarrow \tau][\sigma' \rightarrow \tau']} & & & \uparrow Coe_{[\sigma'][\sigma]} & & \downarrow Coe_{[\tau][\tau']} & \\
Dom_{[\sigma' \rightarrow \tau']} & \cong & [& Dom_{[\sigma']} & \longrightarrow & Dom_{[\tau']} &]
\end{array}$$

$\mathcal{P}_2$: for all $\Gamma \vdash \sigma' \leq \sigma$ and $\Gamma \vdash \tau \leq \tau'$

$$Coe_{[\sigma \rightarrow \tau][\sigma' \rightarrow \tau']} = \Phi^{-1}_{[\sigma' \rightarrow \tau']} \circ FS(Coe_{[\sigma'][\sigma]}, Coe_{[\tau][\tau']}) \circ \Phi_{[\sigma \rightarrow \tau]}$$

If $\mathcal{P}_2$ holds, then (i) and (ii) give the same meaning for $\Gamma \vdash MN : \tau'$.

Now we consider polymorphic types. Let $\Gamma, \alpha : * \vdash f\alpha \leq g\alpha$ so $\Pi f \leq \Pi g$, $\Gamma \vdash \sigma : *$ and $\Gamma \vdash M : \Pi f$. Then $f\sigma \leq g\sigma$, and $\Gamma \vdash M\sigma : g\sigma$ can be derived in several ways, for example:

$$\text{(i)}\quad \dfrac{\dfrac{M : \Pi f \quad \Pi f \leq \Pi g}{M : \Pi g} \quad \sigma : *}{M\sigma : g\sigma} \qquad\qquad \text{(ii)}\quad \dfrac{\dfrac{M : \Pi f \quad \sigma : *}{M\sigma : f\sigma} \quad f\sigma \leq g\sigma}{M\sigma : g\sigma}$$

These two derivations give for $[\![\, \Gamma \vdash M\sigma : g\sigma \,]\!]\eta$

$$(\Phi(\, Coe_{[\Pi f\,][\Pi g\,]}[\![\, \Gamma \vdash M : \Pi f \,]\!]\eta))\ [\![\, \Gamma \vdash \sigma : * \,]\!]\eta \qquad \text{(i)}$$

$$Coe_{[f\sigma\,][g\sigma\,]}((\Phi[\![\, \Gamma \vdash M : \Pi f \,]\!]\eta)\ [\![\, \Gamma \vdash \sigma : * \,]\!]\eta) \qquad \text{(ii)}$$

Of course, we want these to be equal.

Again, there is only one way we can express a relation between $Coe_{[\Pi f\,][\Pi g\,]}$ and $Coe_{[f\sigma\,][g\sigma\,]}$.

$$\begin{array}{lcl} Dom_{[\Pi f\,]} & \cong & \prod_{[\alpha\,]\in Kind_*} Dom_{[f\alpha\,]} \\ \downarrow\ Coe_{[\Pi f\,][\Pi g\,]} & & \downarrow <\ Coe_{[f\alpha\,][g\alpha\,]} \mid [\![\, \alpha \,]\!] \in Kind_* > \\ Dom_{[\Pi g\,]} & \cong & \prod_{[\alpha\,]\in Kind_*} Dom_{[g\alpha\,]} \end{array}$$

$\mathcal{P}_3$: for all $\Gamma, \alpha : * \vdash f\alpha \leq g\alpha$

$$Coe_{[\Pi f\,][\Pi g\,]} = \Phi^{-1}_{[\Pi g\,]} \circ GP(<\ Coe_{[f\alpha\,][g\alpha\,]} \mid [\![\, \alpha \,]\!] \in Kind_* >) \circ \Phi_{[\Pi f\,]}$$

If $\mathcal{P}_3$ holds, then (i) and (ii) do indeed give the same meaning for $\Gamma \vdash M\sigma : g\sigma$.

So we now have the following requirements for the coercion functions

$$\begin{array}{lll}
\mathcal{P}_0 & \text{for all } a \in Kind_* & \\
 & Coe_{a\ a} & = \ \boldsymbol{\lambda}\xi \in Dom_a.\xi \\
\mathcal{P}_1 & \text{for all } a \leq^* b \leq^* c & \\
 & Coe_{a\ b} & = \ Coe_{b\ c} \circ Coe_{a\ b} \\
\mathcal{P}_2 & \text{for all } \Gamma \vdash \sigma \leq \sigma' \text{ and } \Gamma \vdash \tau \leq \tau' & \\
 & Coe_{[\sigma\to\tau\,][\sigma'\to\tau'\,]} & = \ \Phi^{-1}_{[\sigma'\to\tau'\,]} \circ FS(\, Coe_{[\sigma'\,][\sigma\,]},\ Coe_{[\tau\,][\tau'\,]}) \circ \Phi_{[\sigma\to\tau\,]} \\
\mathcal{P}_3 & \text{for all } \Gamma, \alpha : * \vdash f\alpha \leq g\alpha & \\
 & Coe_{[\Pi f\,][\Pi g\,]} & = \ \Phi^{-1}_{[\Pi g\,]} \circ GP(<\ Coe_{[f\alpha\,][g\alpha\,]} \mid [\![\, \alpha \,]\!] \in Kind_* >) \circ \Phi_{[\Pi f\,]}
\end{array}$$

If the coherence conditions $\mathcal{P}_0$, $\mathcal{P}_1$, $\mathcal{P}_2$ and $\mathcal{P}_3$ hold, then the semantics is coherent. In fact, the semantics is coherent if and only if these conditions are satisfied. The proof can be found in the appendix. For the proof we use the fact that we have *minimal typing* in $\Lambda_{\leq}$.

The subtype relation $\leq$ on types induces a subtype relation $\leq^*$ on $Kind_*$.

$$\begin{array}{l} a \leq^* b \\ \text{iff} \\ \exists_{\Gamma,\eta,\sigma,\tau} \Gamma \models \eta\ \&\ [\![\, \Gamma \vdash \sigma : * \,]\!]\eta = a\ \&\ [\![\, \Gamma \vdash \tau : * \,]\!]\eta = b\ \&\ \Gamma \vdash \sigma \leq \tau \end{array}$$

which is the same as

$$\begin{array}{l} [\![\, \Gamma \vdash \sigma : * \,]\!]\eta \leq^* [\![\, \Gamma \vdash \tau : * \,]\!]\eta \\ \text{iff} \\ \Gamma \vdash \sigma \leq \tau \end{array}$$

Because $\leq$ is a pre-order, so is $\leq^*$. Once we have decided on a particular submodel for the constructors, we will give a simpler and more workable definition of $\leq^*$.

So we get the following model definition for $\Lambda_{\leq}$

17 definition (general model definition $\Lambda_{\leq}$)
An environment model for $\Lambda_{\leq}$ is a 7-tuple
$< Kind,\ \Phi_{cons},\ \mathcal{I}_{cons};\ Dom,\ \Phi_{term},\ \mathcal{I}_{term},\ Coe >$,
where Coe is a family of coercion functions satisfying $\mathcal{P}_0$,$\mathcal{P}_1$, $\mathcal{P}_2$ and $\mathcal{P}_3$,

$$Coe =< Coe_{a\ b} \mid \text{for all } a, b \in Kind_* \ ,\ a \leq^* b >$$

where for all $a \leq^* b$

$$Coe_{a\ b} \in [Dom_a \longrightarrow Dom_b]$$

and the rest as in definition 4, with the definition of $[\![\quad]\!]$ for the subsumption rule given by

$$[\![\, \Gamma \vdash M : \tau \,]\!]\eta \ = \ Coe_{[\![\Gamma \vdash \sigma : *]\!]\eta\ [\![\Gamma \vdash \tau : *]\!]\eta}\ [\![\, \Gamma \vdash M : \sigma \,]\!]\eta$$

□

4.3 The construction of a cpo model

We will use the same submodel for the constructor expressions we used for Λ, ie. a term model Because we have a term model we can define $\leq^*$ as follows:

18 definition ($\leq^*$)
If $a, b \in Kind_*$, then a and b are closed type expressions, ie. $<>\vdash a : *$ and $<>\vdash b : *$, and so we can define $\leq^*$ by

$$a \leq^* b \ \text{ iff } \ <>\vdash a \ \leq \ b$$

□

19 lemma $\Gamma \vdash \sigma \leq \tau \iff \forall \eta\, [\![\, \Gamma \vdash \sigma : * \,]\!]\eta \leq^* [\![\, \Gamma \vdash \tau : * \,]\!]\eta$
proof By induction on σ or τ. □

Before we can begin to construct a cpo-model for $\Lambda_{\leq}$, some coercions have to be given. We need coercion functions $coerce_{\sigma\tau}$ from $domain_\sigma$ to $domain_\tau$, for all base types σ and τ such that $\sigma \leq^B \tau$. We require that these coercion functions are continuous, and that $\mathcal{P}_0$ and $\mathcal{P}_1$ hold, ie.

$$\begin{array}{lcll} coerce_{\sigma\sigma} & = & \lambda\xi \in domain_\sigma \ .\ \xi & \\ coerce_{\rho\tau} & = & coerce_{\sigma\tau} \circ coerce_{\rho\sigma} & \text{if } \rho \leq^B \sigma \leq^B \tau \end{array}$$

For $\sigma \leq^B \tau$, $Coe_{\sigma\ \tau} \in [Dom_\sigma \rightarrow Dom_\tau]$ is of course defined by

$$Coe_{\sigma\ \tau} = \Phi_\tau^{-1} \circ coerce_{\sigma\tau} \circ \Phi_\sigma$$

So we are looking for a family of cpos $< Dom_a \mid a \in Kind_* >$, solving the coupled domain equations

$$\begin{array}{rcl} Dom_\sigma & \cong & domain_\sigma \\ Dom_{\sigma\rightarrow\tau} & \cong & FS(Dom_\sigma, Dom_\tau) \\ Dom_{\Pi f} & \cong & GP(< Dom_{fa} \mid a \in Kind_* >) \end{array}$$

and a family of coercion functions $< Coe_{a\ b} \mid a \leq^* b >$ satisfying $\mathcal{P}_0$, $\mathcal{P}_1$ and

$$\begin{array}{lcll} Coe_{\sigma\ \tau} & = & \Phi_\tau^{-1} \circ coerce_{\sigma\tau} \circ \Phi_\sigma & \text{for all } \sigma \leq^B \tau \\ Coe_{\sigma\to\tau\ \sigma'\to\tau'} & = & \Phi_{\sigma'\to\tau'}^{-1} \circ FS(\ Coe_{\sigma'\ \sigma},\ Coe_{\tau\ \tau'}) \circ \Phi_{\sigma\to\tau} & \text{for all } \sigma\to\tau \leq^* \sigma'\to\tau' \\ Coe_{\Pi f\ \Pi g} & = & \Phi_{\Pi g}^{-1} \circ GP(<\ Coe_{fa\ ga} \mid a \in Kind_* >) \circ \Phi_{\Pi f} & \text{for all } \Pi f \leq^* \Pi g \end{array}$$

We define the category corresponding with the subtype relation on $Kind_*$.

20 definition ($\underline{Kind_*}$)
The objects of $\underline{Kind_*}$ are the elements of $Kind_*$, and there is a unique arrow, called $a \leq b$, from a to b iff $a \leq^* b$.
Because $\leq^*$ is reflexive, there is an identity $a \leq a$ for all objects a. Because $\leq^*$ is transitive, composition is always defined: $b \leq c \circ a \leq b$ will be $a \leq c$.
□

Together, *Dom* and *Coe* can be seen as a *functor* from $\underline{Kind_*}$ to $\underline{CPO}$. *Dom* is the object part, mapping every $\underline{Kind_*}$-object, ie. every element of $Kind_*$, to a $\underline{CPO}$-object, a cpo. *Coe* is the morphism part, mapping every $\underline{Kind_*}$-morphism ab 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 $\mathcal{P}_0$ and $\mathcal{P}_1$.
$< domain_\sigma \mid \sigma$ a base type $>$ and $< coerce_{\sigma\tau} \mid \sigma \leq^B \tau >$ form a functor from the category corresponding withthe pre-order $\leq^B$ on base types to $\underline{CPO}$.
We will 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 $[\ \underline{Kind_*},\ \underline{CPO}]$ (usually written $\underline{CPO}^{\underline{Kind_*}}$). Instead, we work in the associated category of embedding-projection pairs. Morphisms of $[\ \underline{Kind_*},\ \underline{CPO}]$ are natural transformations, families of $\underline{CPO}$-morphisms. So pointwise, they have the same properties as $\underline{CPO}$-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 $\underline{CPO}$.

21 definition $[\ \underline{Kind_*},\ \underline{CPO_\perp}]_{PR}$
$[\ \underline{Kind_*},\ \underline{CPO_\perp}]_{PR}$ is the category with as objects functors from $\underline{Kind_*}$ to $\underline{CPO_\perp}$, and as morphisms embedding-projection pairs of natural transformations:
if F and G are functors from $\underline{Kind_*}$ to $\underline{CPO_\perp}$, then (η, θ) is a morphism from F to G if

$$\begin{array}{lcll} \eta & : & F \stackrel{\bullet}{\longrightarrow} G & \text{(ie. } \eta \text{ is a natural transformation from } F \text{ to } G) \\ \theta & : & G \stackrel{\bullet}{\longrightarrow} F & \end{array}$$

and for all $a \in Kind_*$ $\begin{array}{lcl} \theta_a \circ \eta_a & = & id_{Fa} \\ \eta_a \circ \theta_a & \sqsubseteq & id_{Ga} \end{array}$
Composition is of course defined by $(\eta, \theta) \circ (\eta', \theta') = (\eta' \circ \eta, \theta \circ \theta')$
□

The reason for using $\underline{CPO_\perp}$ instead of $\underline{CPO}$, is that $[\,\underline{Kind_*}\,,\ \underline{CPO}]_{PR}$ is not an ω-category, because it does not have an initial element.

22 lemma $[\,\underline{Kind_*}\,,\ \underline{CPO_\perp}]_{PR}$ is an ω-category
proof see [Pol91] □

As a consequence of using $\underline{CPO_\perp}$ instead of $\underline{CPO}$, the coercion functions $coerce_{\sigma\tau}$ will have to be strict. [1] Because $\underline{CPO_\perp}$ is a subcategory of $\underline{CPO}$ and FS and GP preserve strictness, $FS: \underline{CPO_\perp^{OP}} \times \underline{CPO_\perp} \rightarrow \underline{CPO_\perp}$ and $GP: \prod \underline{CPO_\perp} \rightarrow \underline{CPO_\perp}$.
Dom&Coe will be the initial fixed point of the following functor $\mathbb{F}$

23 definition ($\mathbb{F}: \mathbb{K} \rightarrow \mathbb{K}$)
$\mathbb{F}$ is a functor $\mathbb{K}$ to $\mathbb{K}$, so it consists of an object part, a mapping from $Obj(\mathbb{K})$ to $Obj(\mathbb{K})$, and an morphism part, a mapping from $Mor(\mathbb{K})$ to $Mor(\mathbb{K})$.
The object part of $\mathbb{F}$ is defined as follows. Let $F \in Obj(\mathbb{K})$. Then $\mathbb{F}F \in Obj(\mathbb{K})$, ie. $\mathbb{F}F$ is a functor from $\underline{Kind_*}$ to $\underline{CPO_\perp}$.
The object part of $\mathbb{F}F$, a mapping from $Obj(\,\underline{Kind_*}\,)$ to $Obj(\,\underline{CPO_\perp}\,)$, is defined by

$$\begin{array}{lcl} (\mathbb{F}F)a & = & domain_a \\ (\mathbb{F}F)a \rightarrow b & = & FS(Fa, Fb) \\ (\mathbb{F}F)\Pi f & = & GP(< F(fa) \mid a \in Kind_* >) \end{array}$$

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

$$\begin{array}{lcl} (\mathbb{F}F)a \leq b & = & coerce_{ab} \\ (\mathbb{F}F)a \rightarrow b \leq a' \rightarrow b' & = & FS(Fa' \leq a, Fb \leq b') \\ (\mathbb{F}F)\Pi f \leq \Pi g & = & GP(< Ffa \leq ga \mid a \in Kind_* >) \end{array}$$

The morphism part of $\mathbb{F}$ is defined as follows:
if $(\eta, \theta) \in Hom_{\mathbb{K}}(F, G)$, so $\eta : F \xrightarrow{\bullet} G$ and $\theta : G \xrightarrow{\bullet} F$ then
$\mathbb{F}((\eta, \theta)) = (\eta', \theta')$, ie. $\eta' : \mathbb{F}F \xrightarrow{\bullet} \mathbb{F}G$ and $\theta' : \mathbb{F}G \xrightarrow{\bullet} \mathbb{F}F$ where

$$\begin{array}{lcl} (\eta'_a, \theta'_a) & = & (id_{domain_a}, id_{domain_a}) \\ (\eta'_{a \rightarrow b}, \theta'_{a \rightarrow b}) & = & FS_{PR}((\eta_a, \theta_a), (\eta_b, \theta_b)) \\ (\eta'_{\Pi f}, \theta'_{\Pi f}) & = & GP_{PR}(< (\eta_{fa}, \theta_{fa}) \mid a \in Kind_* >) \end{array}$$

Checking $\eta' : \mathbb{F}F \xrightarrow{\bullet} \mathbb{F}G$ and $\theta' : \mathbb{F}G \xrightarrow{\bullet} \mathbb{F}F$ is straightforward, and it can easily be verified (pointwise) that $\mathbb{F}$ 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

$$\frac{\Gamma \vdash \sigma' \leq \sigma \quad \Gamma \vdash \tau \leq \tau'}{\Gamma \vdash \sigma \rightarrow \tau \leq \sigma' \rightarrow \tau'}$$

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 $\mathbb{F}$ will solve the recursive domain equations and satisfy the conditions for the coercion functions.

[1] The requirement that the coercions be strict also comes up in [BTCGS89] , although for different reasons.

For instance, let $(F,(\Phi,\Psi))$ be a fixed point of $\mathbb{F}$, ie. (Φ,Ψ) is an isomorphism between F and $\mathbb{F}F$. This means that $\Phi : F \stackrel{\bullet}{\longrightarrow} \mathbb{F}F$ and $\Psi : \mathbb{F}F \stackrel{\bullet}{\longrightarrow} F$, such that $\Phi \circ \Psi = id_{\mathbb{F}F}$ and $\Psi \circ \Phi = id_F$. Because everything is defined pointwise, this means that for all $a \leq^* b$

$$\begin{array}{rcl} \Phi_b \circ \Psi_b & = & id_{(\mathbb{F}F)b} \\ \Psi_b \circ \Phi_b & = & id_{Fb} \\ \Phi_a \circ \Psi_a & = & id_{(\mathbb{F}F)a} \\ \Psi_a \circ \Phi_a & = & id_{Fa} \end{array} \quad \text{and} \quad \begin{tikzcd} Fa \arrow[r, shift left, "\Phi_a"] \arrow[d, "Fa \leq b"'] & (\mathbb{F}F)a \arrow[l, shift left, "\Psi_a"] \arrow[d, "(\mathbb{F}F)a \leq b"] \\ Fb \arrow[r, shift left, "\Phi_b"] & (\mathbb{F}F)b \arrow[l, shift left, "\Psi_b"] \end{tikzcd}$$

Let $\Pi f \leq^* \Pi g$. Then

$$\begin{tikzcd} F\Pi f \arrow[r, shift left, "\Phi_{\Pi f}"] \arrow[d, "F\Pi f \leq \Pi g"'] & (\mathbb{F}F)\Pi f = GP(< F(fa) \mid a \in Kind_* >) \arrow[l, shift left, "\Psi_{\Pi f}"] \arrow[d, "(\mathbb{F}F)\Pi f \leq \Pi g \; = GP(< F(fa)(ga) \mid a \in Kind_* >)"] \\ F\Pi g \arrow[r, shift left, "\Phi_{\Pi g}"] & (\mathbb{F}F)\Pi g \; = GP(< F(ga) \mid a \in Kind_* >) \arrow[l, shift left, "\Psi_{\Pi g}"] \end{tikzcd}$$

$$\begin{array}{rcl} \text{and} \quad F\Pi g \leq \Pi f & = & \Psi_{\Pi g} \circ ((\mathbb{F}F)\Pi f \leq \Pi g) \circ \Phi_{\Pi f} \\ & = & \Psi_{\Pi g} \circ GP(< Ffa \leq ga \mid a \in Kind_* >) \circ \Phi_{\Pi f} \end{array}$$

so $\mathcal{P}_3$ is satisfied. In the same way it can be shown that $\mathcal{P}_2$ is satisfied.

24 lemma $\mathbb{F}$ is ω-continuous

proof (sketch,for details see [Pol91])

We define a functor $\mathcal{H}$: $[\,\underline{Kind_*}\,,\ \underline{CPO}]^{OP} \times [\,\underline{Kind_*}\,,\ \underline{CPO}] \rightarrow [\,\underline{Kind_*}\,,\ \underline{CPO}]$ such that

$$\begin{array}{rclcl} \mathbb{F}F & = & \mathcal{H}(F,F) & = & \mathcal{H}_{PR}(F,F) \\ \mathbb{F}(\eta,\theta) & = & (\mathcal{H}(\theta,\eta),\mathcal{H}(\eta,\theta)) & = & \mathcal{H}_{PR}((\eta,\theta),(\eta,\theta)) \end{array}$$

We prove so-called local continuity for $\mathcal{H}$, which can be done pointwise. This means that $\mathcal{H}_{PR}$ is ω-continuous. Using the correspondence between $\mathbb{F}$ and $\mathcal{H}_{PR}$ given above, we can prove that if $\mathcal{H}_{PR}$ is ω-continuous, $\mathbb{F}$ is also ω-continuous.

□

So by the initial fixed point theorem $\mathbb{F}$ has an initial solution $(Dom\&Coe,(\Phi,\Psi))$. The object part of $Dom\&Coe$ $\mathbb{F}$ 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
- $[\,\underline{Kind_*}\,,\ \underline{CPO}_\perp]$ is an O-category
- $[\,\underline{Kind_*}\,,\ \underline{CPO}_\perp]_{PR}$ is an ω-category

- $\mathbb{F}$ is ω-continuous
- in $[\ \underline{Kind_*},\ \underline{CPO_\perp}\ ]_{PR}$ the equation $\mathbb{F}(D) \cong D$ has an initial solution $(Dom\&Coe, (\Phi, \Psi))$
- $(Dom\&Coe, (\Phi, \Psi))$ 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 will now combine the two extensions of Λ we have dealt with, subtyping and recursive types.

5.1 Syntax

First we consider how to define the subtype relation on recursive types. The natural rule for subtyping on recursive types is

$$\frac{\Gamma, \alpha : *, \beta : *, \alpha \leq \beta \vdash f\alpha \leq g\beta}{\Gamma \vdash \mu f \leq \mu g}(\leq \mu)$$

This is the same as

$$\frac{\Gamma, \alpha : * \vdash f\alpha \leq g\alpha}{\Gamma \vdash \mu f \leq \mu g}$$

where α may only occurs at covariant positions in $f\alpha$ or $g\alpha$.

Contexts can now also also contain expressions of the form $\alpha \leq \beta$, where α and β are type variables, but only when we are deriving subtype judgements. In $\Gamma \vdash M : \sigma$ the contex will not contain expressions of the form $\alpha \leq \beta$.

We will now also need the rule

$$\Gamma, \alpha \leq \beta \vdash \alpha \leq \beta$$

Since we considered three ways to incorporate recursive types in Λ, several options are open to us. The systems we get by extending $\Lambda\mu_1$,$\Lambda\mu_2$ and $\Lambda\mu_3$ with subtyping will be called $\Lambda{\leq}\mu_1$,$\Lambda{\leq}\mu_2$ and $\Lambda{\leq}\mu_3$, respectively.

Since in $\Lambda\mu_1$ $\mu f \neq f(\mu f)$, we could add the following rules for $\Lambda{\leq}\mu_1$

$$\frac{\Gamma \vdash f : * \Rightarrow *}{\Gamma \vdash \mu f \leq f(\mu f)} \qquad \frac{\Gamma \vdash f : * \Rightarrow *}{\Gamma \vdash f(\mu f) \leq \mu f}$$

The $fold_{\mu f}$ and $unfold_{\mu f}$ can then be omitted. The coercions for $\mu f \leq f(\mu f)$ and $f(\mu f) \leq \mu f$ are of course $\Phi_{\mu f}$ and $\Phi_{\mu f}^{-1}$. However, the resulting system is then virtually the same as $\Lambda{\leq}\mu_2$, because the same type derivations $\Gamma \vdash M : \sigma$ will be derivable. The only difference is the notion of constructor equality.

5.2 Semantics : general model definition

Remember that we can now have expressions such as $\alpha \leq \beta$ in contexts. For an environment η to satisfy a context Γ we now also require that

$$\eta(\alpha) \leq^* \eta(\beta) \qquad \text{for all } (\alpha \leq \beta) \in \Gamma$$

We get environment models for these systems with subtyping and recursive types by extending a model for the corresponding system without subtyping with a family of coercion functions $Coe = < Coe_{a\,b} \mid a, b \in Kind_* \ , \ a {\leq^*} b >$

25 definition (general model definition $\Lambda{\leq}\mu_1$, $\Lambda{\leq}\mu_2$ and $\Lambda{\leq}\mu_3$)
A second order environment model for $\Lambda{\leq}\mu_1$, $\Lambda{\leq}\mu_2$ or $\Lambda{\leq}\mu_3$ is a 7-tuple
$< Kind, \Phi_{cons}, \mathcal{I}_{cons}; Dom, \Phi_{term}, \mathcal{I}_{term}, Coe >$,
where Coe is a family of coercion functions,

$$Coe = < Coe_{a\,b} \in [Dom_a \longrightarrow Dom_b] \mid a, b \in Kind_* \ , \ a {\leq^*} b >$$

satisfying $\mathcal{P}_0$, $\mathcal{P}_1$, Parrow and $\mathcal{P}_3$, and the rest as in the definition of the general model definition for $\boldsymbol{\Lambda\mu_1}$, $\boldsymbol{\Lambda\mu_2}$ or $\boldsymbol{\Lambda\mu_3}$ (definitions 5,6 and 10).
□

26 theorem (coherence)
The semantics of $\Lambda{\leq}\mu_1$, $\Lambda{\leq}\mu_2$ and $\Lambda{\leq}\mu_3$ are coherent
proof
For the systems $\Lambda{\leq}\mu_2$ and $\Lambda{\leq}\mu_3$ we have the same type inference rules as for $\Lambda_{\leq}$. So the proof of coherence for $\Lambda_{\leq}$ (theorem 39) also proves coherence for $\Lambda{\leq}\mu_2$ and $\Lambda{\leq}\mu_3$.
The two extra type inference rules that we have in $\Lambda{\leq}\mu_1$, viz. $(FOLD)$ and $(UNFOLD)$

$$\frac{\Gamma \vdash M : \mu f}{\Gamma \vdash \ unfold_{\mu f} M : f(\mu f)}\ (UNFOLD) \qquad \frac{\Gamma \vdash M : f(\mu f)}{\Gamma \vdash \ fold_{\mu f} M : \mu f}\ (FOLD)$$

do not pose a problem as far as coherence is concerned, because of the subscripts of $fold_{\mu f}$ and $unfold_{\mu f}$.
□

For the model constructions we only have to define a pre-order on $Kind_*$ that corresponds with the subtype relation on types. We can then construct a model in the same way as we did for $\Lambda_{\leq}$, as an initial fixed point of a functor $\mathbb{F}$ on $[\ \underline{Kind_*},\ \underline{CPO_\perp}]_{PR}$.

5.3 The construction of a cpo model for $\Lambda{\leq}\mu_1$

For the model construction we will again need some properties of the subtype relation:

27 lemma
$$\begin{array}{rcl} \Gamma \vdash \sigma \to \tau \ \leq \ \sigma' \to \tau' & \Longrightarrow & \Gamma \vdash \sigma' \leq \sigma \text{ and } \tau \leq \tau' \\ \Gamma \vdash \Pi f \ \leq \ \Pi g & \Longrightarrow & \Gamma, \alpha : * \vdash f\alpha \leq g\alpha \\ \Gamma \vdash \mu f \ \leq \ \mu g & \Longrightarrow & \Gamma, \alpha : *, \beta : *, \alpha \leq \beta \vdash f\alpha \leq g\beta \end{array}$$
□

We prove this in the same way as we proved lemma 14 . We define a relation $\leq'$ on types. For $\leq'$ we have the same derivation rules as for $\leq$, except instead of $(TRANS)$ we have the rule $(\leq TEQ)$. Clearly $\Gamma \vdash \sigma \leq' \tau \ \Rightarrow\ \Gamma \vdash \sigma \leq \tau$, and by the next lemma we also have $\Gamma \vdash \sigma \leq \tau \ \Rightarrow\ \Gamma \vdash \sigma \leq' \tau$.

28 lemma $\leq'$ is transitive, i.e. $\Gamma \vdash \rho \leq' \sigma \ \& \Gamma \vdash \sigma \leq' \tau \ \Rightarrow\ \Gamma \vdash \rho \leq' \tau$
proof
The proof is almost the same as for lemma 15. The only difference is that we now also have the possibility that

(d) $\rho =_c \mu f$, $\sigma =_c \mu g$ and $\tau =_c \mu h$.

For this case $\Gamma \vdash \rho \leq' \tau$ is proven as for (b) and (c):
The derivations of $\Gamma \vdash \rho \leq' \sigma$ and $\Gamma \vdash \sigma \leq' \tau$, must both end with $(\leq\mu)$, possibly followed by $(\leq TEQ)$. So $\Gamma, \alpha : *, \beta : *, \alpha{\leq}\beta \vdash f\alpha \leq' g\beta$ and $\Gamma, \beta : *, \gamma : *, \beta{\leq}\gamma \vdash g\beta \leq' h\gamma$.
By the induction hypothesis $\Gamma, \alpha : *, \gamma : *, \alpha{\leq}\gamma \vdash h\alpha \leq' h\gamma$, so $\Gamma \vdash \mu f \leq' \mu h$ and hence $\Gamma \vdash \rho \leq' \tau$.
□

So $\Gamma \vdash \sigma \leq' \tau \ \Leftrightarrow\ \Gamma \vdash \sigma \leq \tau$, and for $\leq'$ it is obvious that lemma 27 holds.
We will also need

29 lemma $\Gamma \vdash \mu f \leq \mu g \;\; \Rightarrow \;\; \Gamma \vdash f(\mu f) \leq g(\mu g)$
proof
Suppose $\Gamma \vdash \mu f \leq \mu g$.
Then $\Gamma, \alpha : *, \beta : *, \alpha \leq \beta \vdash f\alpha \leq g\beta$, and in the derivation of this we can substitute μf for α and μg for β, which gives us $\Gamma \vdash f(\mu f) \leq g(\mu g)$
□

We again use a term model as the submodel for the constructor expressions. We define the relation $\leq^*$ on $Kind_*$ as we did for the model construction for $\Lambda_{\leq}$.

30 definition ($\leq^*$)
If $a, b \in Kind_*$, then a and b are closed type expressions, i.e. $<>\vdash a : *$ and $<>\vdash b : *$, so we can define $\leq^*$by

$$a \leq^* b \text{ iff } <>\vdash a \leq b$$

□

31 lemma $\Gamma \vdash \sigma \leq \tau \iff \forall_\eta [\![\Gamma \vdash \sigma : *]\!] \eta \leq^* [\![\Gamma \vdash \tau : *]\!] \eta$
proof By induction on σ or τ. □

We define a functor $\mathbb{F}$ on $\mathbb{K}$, $\mathbb{K} = [\, \underline{Kind_*},\ \underline{CPO_\perp}]_{PR}$.

32 definition ($\mathbb{F} : \mathbb{K} \to \mathbb{K}$)
The object part of $\mathbb{F}$ is defined as follows. Let $F \in Obj(\mathbb{K})$. Then $\mathbb{F}F \in Obj(\mathbb{K})$, i.e. $\mathbb{F}F$ is a functor from $\underline{Kind_*}$ to $\underline{CPO_\perp}$. The object part of $\mathbb{F}F$, is defined by

$$\begin{array}{lcl} (\mathbb{F}F)a & = & domain_a \\ (\mathbb{F}F)\sigma \to \tau & = & FS(F\sigma, F\tau) \\ (\mathbb{F}F)\Pi f & = & GP(< F(fa) \mid a \in Kind_* >) \\ (\mathbb{F}F)\mu f & = & F(f(\mu f)) \end{array}$$

and the morphism part of $\mathbb{F}F$, is defined by

$$\begin{array}{lcl} (\mathbb{F}F)a{\leq}b & = & coerce_{ab} \\ (\mathbb{F}F)\sigma \to \tau{\leq}\sigma' \to \tau' & = & FS(F\ \sigma'{\leq}\sigma, F\ \tau{\leq}\tau') \\ (\mathbb{F}F)\Pi f{\leq}\Pi g & = & GP(< F\ fa{\leq}ga \mid a \in Kind_* >) \\ (\mathbb{F}F)\mu f{\leq}\mu g & = & F\ f(\mu f){\leq}g(\mu g) \end{array}$$

The morphism part of $\mathbb{F}$ is defined as follows:
if $(\eta, \theta) \in Hom_{\mathbb{K}}(F, G)$, so $\eta : F \stackrel{\bullet}{\longrightarrow} G$ and $\theta : G \stackrel{\bullet}{\longrightarrow} F$ then $\mathbb{F}(\eta, \theta) = (\eta', \theta')$, where

$$\begin{array}{lcl} (\eta'_a, \theta'_a) & = & (id_{domain_a}, id_{domain_a}) \\ (\eta'_{\sigma\to\tau}, \theta'_{\sigma\to\tau}) & = & FS_{PR}((\eta_\sigma, \theta_\sigma), (\eta_\tau, \theta_\tau)) \\ (\eta'_{\Pi f}, \theta'_{\Pi f}) & = & GP_{PR}(< (\eta_{fa}, \theta_{fa}) \mid a \in Kind_* >) \\ (\eta'_{\mu f}, \theta'_{\mu f}) & = & (\eta_{f(\mu f)}, \theta_{f(\mu f)}) \end{array}$$

That $\mathbb{F}$ preserves identities and composition can easily be verified (pointwise).
□

In the same way lemma 24 is proved, we can prove that $\mathbb{F}$ is ω-continuous.
So $\mathbb{F}$ has an initial fixed point $(Dom\&Coe, (\Phi, \Psi))$, 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.

For the coercions between recursive types

$$Coe_{\mu f\ \mu g} \quad = \quad \Phi_{\mu g}^{-1} \circ Coe_{f(\mu f)\ g(\mu g)} \circ \Phi_{\mu f} \qquad (*)$$

will hold. This means that coercions commute with unfolding and folding, i.e. $Coe_{\mu f\ \mu g}$ followed by $\mathit{unfold}_{\mu g}$ gives the same result as $\mathit{unfold}_{\mu f}$ followed by $Coe_{f(\mu f)\ g(\mu g)}$, and $\mathit{fold}_{\mu f}$ followed by $Coe_{\mu f\ \mu g}$ gives the same result as $Coe_{f(\mu f)\ g(\mu g)}$ followed by $\mathit{fold}_{\mu g}$. However, because of the subscripts of *fold* and *unfold* this is *not* needed for coherence; $Coe_{\mu f\ \mu g}$ and $Coe_{f(\mu f)\ g(\mu g)}$ could be completely unrelated.
Because of $(*)$, the subscript of *unfold* can be omitted. We can check that lemma 38 holds for the rule $(UNFOLD)$, so the semantics will then still be coherent.
Possibly the subscript of *fold* can also be omitted. However, as shown in the example on page 16, there are two possible types for a term *fold* M, and *fold* M does not have a *minimal* type. Therefore the coherence proof as given in the appendix can not be used.

5.4 The construction of a cpo model for $\Lambda{\leq}\mu_3$

We distinguish *covariant* and *contravariant* positions in trees. A node or leaf in t is at a covariant position in t if, going from that node or leaf to the root of t, we enter an even number of $\rightarrow$-nodes from the left-hand side, and else it is at a contravariant position in t. For example, in

$$\begin{tikzcd}[column sep=small] & & \longrightarrow \arrow[dl] \arrow[dr] & \\ & \longrightarrow \arrow[dl] \arrow[dr] & & \alpha \\ int & & real & \end{tikzcd}$$

int and α occur at covariant positions, whereas *real* and $\begin{tikzcd}[column sep=small] & \longrightarrow \arrow[dl] \arrow[dr] & \\ int & & real \end{tikzcd}$ occur at contravariant positions.

33 definition $(\leq^*)$
$s \leq^* t$
iff
except for their leaves, s and t are the same tree, and for all leaves a and b in the same place in s and t, respectively:

- $a \leq^B b$ and a and b occur at covariant positions in s and t , or
- $b \leq^B a$ and a and b occur at contravariant positions in s and t , or
- $a \equiv b$

□

We want to prove

$$\Gamma \vdash \sigma \leq \tau \iff \forall \eta \, [\![\, \Gamma \vdash \sigma : * \,]\!] \eta \leq^* [\![\, \Gamma \vdash \tau : * \,]\!] \eta$$

It is really the implication $\Longrightarrow$ that is important, since if that implication holds, then a family of coercion functions
$\quad < \; Coe_{a\;b} \mid a \leq^* b >$
will contain the required coercions.

34 lemma $\Gamma \vdash \sigma \leq \tau \implies \forall_{\Gamma \models \eta} \; [\![\, \Gamma \vdash \sigma : * \,]\!] \eta \leq^* [\![\, \Gamma \vdash \tau : * \,]\!] \eta$
proof by induction on the derivation of $\Gamma \vdash \sigma \leq \tau$.
We only treat the prime case, $(\leq \mu)$. Suppose the last rule of the derivation is $(\leq \mu)$,

$$\frac{\Gamma, \alpha : *, \beta : *, \alpha \leq \beta \vdash f\alpha \leq g\beta}{\Gamma \vdash \mu f \leq \mu g}$$

Define $\Gamma' = \Gamma, \alpha : *, \beta : *, \alpha \leq \beta$.
By the induction hypothesis : $\forall_{\Gamma' \models \eta} \; [\![\, \Gamma' \vdash f\alpha : * \,]\!] \eta \leq^* [\![\, \Gamma' \vdash g\beta : * \,]\!] \eta$.
To prove : $\forall_{\Gamma \models \eta} \; [\![\, \Gamma \vdash \mu f : * \,]\!] \eta \leq^* [\![\, \Gamma \vdash \mu g : * \,]\!] \eta$.
Assume $\Gamma \models \eta$. Define $F = [\![\, \Gamma \vdash f : * \Rightarrow * \,]\!] \eta$ and $G = [\![\, \Gamma \vdash g : * \Rightarrow * \,]\!] \eta$.
By induction on $i \in \mathbb{N}$ we prove $F^i \bot \leq^* G^i \bot$.

base $F^0 \bot = \bot \leq^* \bot = G^0 \bot$

step $F^{i+1}\bot = ([\![\,\Gamma \vdash f : * \Rightarrow *\,]\!]\eta)(F^i\bot) = [\![\,\Gamma' \vdash f\alpha : *\,]\!]\eta[\alpha := F^i\bot][\beta := G^i\bot]$
$G^{i+1}\bot = ([\![\,\Gamma \vdash g : * \Rightarrow *\,]\!]\eta)(G^i\bot) = [\![\,\Gamma' \vdash g\beta : *\,]\!]\eta[\alpha := F^i\bot][\beta := G^i\bot]$
$F^i\bot \le^* G^i\bot$, so $\eta[\alpha := F^i\bot][\beta := G^i\bot]$ satisfies Γ'. Then by the induction hypothesis
$[\![\,\Gamma' \vdash f\alpha : *\,]\!]\eta[\alpha := F^i\bot][\beta := G^i\bot] \le^* [\![\,\Gamma' \vdash g\beta : *\,]\!]\eta[\alpha := F^i\bot][\beta := G^i\bot]$
so $F^{i+1}\bot \le^* G^{i+1}\bot$.

$[\![\,\Gamma \vdash \mu f : *\,]\!]\eta \;=\; \bigsqcup F^i\bot \le^* \bigsqcup G^i\bot \;=\; [\![\,\Gamma \vdash \mu g : *\,]\!]\eta.$
□

It is easy to see that

$$\begin{tikzcd}[column sep=tiny] & \longrightarrow \arrow[dl] \arrow[dr] & & \le^* & & \longrightarrow \arrow[dl] \arrow[dr] & & \Longrightarrow & s' \le^* s \,\wedge\, t \le^* t' \\ s & & t & & s' & & t' & & \end{tikzcd}$$

$$\begin{tikzcd}[column sep=small] \Pi_\alpha \arrow[d] & \le^* & \Pi_\alpha \arrow[d] & \Longrightarrow & s \le^* t \\ s & & t & & \end{tikzcd}$$

so

$$[\![\,\sigma \to \tau\,]\!] \le^* [\![\,\sigma' \to \tau'\,]\!] \;\Longrightarrow\; [\![\,\sigma'\,]\!] \le^* [\![\,\sigma\,]\!] \text{ and } [\![\,\tau\,]\!] \le^* [\![\,\tau'\,]\!]$$
$$[\![\,\Pi f\,]\!] \le^* [\![\,\Pi g\,]\!] \;\Longrightarrow\; \forall_{[\![\,\alpha\,]\!] \in Kind_*}\, [\![\,f\alpha : *\,]\!] \le^* [\![\,g\alpha : *\,]\!]$$

By $\mathcal{P}_2$: for all $[\![\,\sigma \to \tau\,]\!] \le^* [\![\,\sigma' \to \tau'\,]\!]$

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

and by $\mathcal{P}_3$: for all $[\![\,\Pi f\,]\!] \le^* [\![\,\Pi g\,]\!]$

$$Coe_{[\![\,\Pi f\,]\!][\![\,\Pi g\,]\!]} \;=\; \Phi^{-1}_{[\![\,\Pi g\,]\!]} \circ GP(<\; Coe_{[\![\,f\alpha\,]\!][\![\,g\alpha\,]\!]} \mid [\![\,\alpha\,]\!] \in Kind_* >) \circ \Phi_{[\![\,\Pi f\,]\!]}$$

To construct the required family of cpos and a family of coercion functions we can now use the same construction we used for $\Lambda_{\le}$.

35 definition ($\mathbb{F} : \mathbb{K} \to \mathbb{K}$)
The object part of $\mathbb{F}$ is defined as follows. Let $F \in Obj(\mathbb{K})$. Then the object part of $\mathbb{F}F$, a mapping from $Obj(\,\underline{Kind_*}\,)$ to $Obj(\,\underline{CPO_\bot}\,)$, is defined by

$$\begin{array}{lcl} (\mathbb{F}F).\sigma & = & domain_\sigma \\ (\mathbb{F}F)\; \begin{tikzcd}[column sep=tiny] & \longrightarrow \arrow[dl] \arrow[dr] & \\ \sigma & & \tau \end{tikzcd} & = & FS(F\sigma, F\tau) \\ (\mathbb{F}F)\; \begin{tikzcd} \Pi_\alpha \arrow[d] \\ \tau \end{tikzcd} & = & GP(< F(\tau[\alpha := a]) \mid a \in Kind_* >) \\ (\mathbb{F}F)\bot & = & F\bot \end{array}$$

and the morphism part of $\mathbb{F}F$, a mapping from $Mor(\,\underline{Kind_*}\,)$ to $Mor(\,\underline{CPO_\bot}\,)$, is defined by

$$\begin{array}{lcl} (\mathbb{F}F).\sigma \le .\tau & = & coerce_{\sigma\tau} \\ (\mathbb{F}F)\; \begin{tikzcd}[column sep=tiny] & \longrightarrow \arrow[dl] \arrow[dr] & & \le & & \longrightarrow \arrow[dl] \arrow[dr] & \\ \sigma & & \tau & & \sigma' & & \tau' \end{tikzcd} & = & FS(F\sigma' \le \sigma, F\tau' \le \tau) \\ (\mathbb{F}F)\; \begin{tikzcd}[column sep=small] \Pi_\alpha \arrow[d] & \le & \Pi_\alpha \arrow[d] \\ \sigma & & \tau \end{tikzcd} & = & GP(< F\sigma[\alpha := a] \le \tau[\alpha := a] \;\mid a \in Kind_* >) \\ (\mathbb{F}F)\bot \le \bot & = & F\,\bot \le \bot \end{array}$$

The morphism part of $I\!\!F$ is defined as follows:
if $(\eta,\theta) \in Hom_{I\!\!K}(F,G)$, then $I\!\!F(\eta,\theta) = (\eta',\theta')$, where

$$
\begin{array}{lcl}
(\eta'_{\sigma}, \theta'_{\sigma}) & = & (id_{domain_\sigma}, id_{domain_\sigma}) \\
(\eta'_{\substack{\longrightarrow \\ \swarrow \quad \searrow \\ \sigma \qquad \tau}}, \theta'_{\substack{\longrightarrow \\ \swarrow \quad \searrow \\ \sigma \qquad \tau}}) & = & FS_{PR}((\eta_\sigma,\theta_\sigma),(\eta_\tau,\theta_\tau)) \\
(\eta'_{\substack{\Pi_\alpha \\ \downarrow \\ \sigma}}, \theta'_{\substack{\Pi_\alpha \\ \downarrow \\ \sigma}}) & = & GP_{PR}(< \ (\eta_{\sigma[\alpha:=a]}, \theta_{\sigma[\alpha:=a]}) \ \mid a \in Kind_* >) \\
(\eta'_\perp, \theta'_\perp) & = & (\eta_\perp, \theta_\perp)
\end{array}
$$

□

In the same way lemma 24 is proved, we can prove that $I\!\!F$ is ω-continuous.
So $I\!\!F$ has an initial fixed point $(Dom\&Coe, (\Phi,\Psi))$ which gives us a family of cpos solving the recursive domain equations with the asociated bijections , and a family of coercions satisfying the coherence conditions.

5.5 The construction of a cpo model for $\Lambda\leq\mu_2$

The only difficulty for $\Lambda\leq\mu_2$ is that to construct a model we have to prove

$$
\begin{array}{rcl}
\Gamma \vdash \sigma \to \tau \ \leq \ \sigma' \to \tau' & \Longrightarrow & \Gamma \vdash \sigma' \leq \sigma \text{ and } \tau \leq \tau' \\
\Gamma \vdash \Pi f \ \leq \ \Pi g & \Longrightarrow & \Gamma, \alpha : * \vdash f\alpha \leq g\alpha
\end{array}
$$

which is still an open problem.
Once it is proved, we can construct a model in the same way the models for $\Lambda\leq\mu_1$ and $\Lambda\leq\mu_3$ have been constructed.
The subtype relation on $Kind_*$ is of course defined as it was for $\Lambda\leq\mu_1$(definition 30).

6 Conclusion

The theory of O-categories has proved extremely useful. 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 O-category $\underline{CPO}$ is not essential. Other O-categories could be used, for instance the category of directed cpos or complete lattices: types would then be interpreted as directed cpos or complete lattices.

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 included. We add them as constructor constants of the approprate kind, and add the associated type inference rules. 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 $\rightarrow$-types.

For example, for $\times$-types we would have to add a type constructor $\times$ of kind $* \Rightarrow *(* \Rightarrow *)$ and the recursive domain equations

$$Dom_{[\![\sigma\times\tau]\!]} \;\cong\; Dom_{[\![\sigma]\!]} \times Dom_{[\![\sigma]\!]}$$

so we would have to extend the definition of F with

$$F_{[\![\sigma\times\tau]\!]}(< D_a \mid a \in Kind_* >) \;=\; CP(D_{[\![\sigma]\!]}, D_{[\![\tau]\!]})$$

where CP is the product functor. The natural subtyping rule for $\times$-types

$$\frac{\Gamma \vdash \sigma \leq \sigma' \quad \Gamma \vdash \tau \leq \tau'}{\Gamma \vdash \sigma \times \tau \leq \sigma' \times \tau'}$$

can be added, and for coherence we will need the additional requirement

$$Coe_{[\![\sigma\times\tau]\!][\![\sigma'\times\tau']\!]} \;=\; \Phi^{-1}_{\sigma'\times\tau'} \circ CP(\,Coe_{[\![\sigma]\!][\![\sigma']\!]},\, Coe_{[\![\tau]\!][\![\tau']\!]}) \circ \Phi_{\sigma\times\tau}$$

The type constructor Σ, which can be used for abstract data types (see [MP88]), can also be added. These Σ-types or existential types, can be treated like the Π-types. Just like the generalized product functor is used for Π-types, the generalized sum functor (see [tEH89b]) can be used for Σ-types.

For the systems with subtyping, interesting extensions are of course labelled products, i.e. records, and bounded quantification.
For bounded quantification we have the type formation rule

$$\frac{\Gamma, \alpha : *, \alpha \leq \sigma \vdash \tau : * \qquad \Gamma \vdash \sigma : *}{\Gamma \vdash (\Pi\alpha{\leq}\sigma.\tau) : *}$$

The recursive domain equations for such a type is

$$Dom_{[\![\Gamma\vdash(\Pi\alpha\leq\sigma.\tau):*]\!]\eta} \;\cong\; \prod_{a\in Kind_*, a\leq^*[\![\Gamma\vdash\sigma:*]\!]\eta} Dom_{[\![\Gamma,\alpha:*\vdash\tau:*]\!]\eta[\alpha:=a]}$$

so we get

$$Dom_{[\![(\Pi\alpha\leq\sigma.\tau)]\!]} \;\cong\; GP(< Dom_{[\![\tau]\!]} \mid [\![\alpha]\!] \in Kind_*\,,\ [\![\alpha]\!] \leq^* [\![\sigma]\!] >)$$

The subtyping rule for Π-types becomes

$$\frac{\Gamma,\ \alpha:*,\ \alpha\leq\sigma\ \vdash\ \tau\leq\tau' \quad \Gamma\vdash\sigma'\leq\sigma}{\Gamma\vdash(\Pi\alpha\leq\sigma.\tau)\ \leq\ (\Pi\alpha\leq\sigma'.\tau')}(\leq\Pi)$$

and for the coercion functions we get the following coherence conditions

$$\begin{array}{ll} & Coe_{[\![(\Pi\alpha\leq\sigma.\tau)]\!][\![(\Pi\alpha\leq\sigma'.\tau')]\!]} \\ = & \\ & \Phi^{-1}_{(\Pi\alpha\leq\sigma'.\tau')}\circ GP(<\ Coe_{[\![\tau]\!][\![\tau']\!]}\mid[\![\alpha]\!]\leq^*[\![\sigma']\!]>)\circ<proj_a\mid a\in Kind_*, a\leq^*[\![\sigma']\!]>\circ\Phi_{(\Pi\alpha\leq\sigma.\tau)}\end{array}$$

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

$$proj_a\ \in\ (\prod_{a\leq^*[\![\sigma]\!]} Dom_{[\![\tau]\!]\eta[\alpha:=a]}) \longrightarrow Dom_{[\![\tau]\!]\eta[\alpha:=a]}$$

and

$$<proj_a\mid a\leq^*[\![\sigma]\!]>\ \in\ (\prod_{a\leq^*[\![\sigma]\!]} Dom_{[\![\tau]\!]\eta[\alpha:=a]}) \longrightarrow (\prod_{a\leq^*[\![\sigma']\!]} Dom_{[\![\tau]\!]\eta[\alpha:=a]})$$

Labelled products can be handled similarly. For these types we have the type formation rule

$$\frac{\Gamma\vdash\sigma_1:*,\ldots,\sigma_n:* \quad l_1,\ldots,l_n\in\mathcal{L} \quad \forall_{i,j}(l_i=l_j\Rightarrow i=j)}{\Gamma\vdash<l_1:\sigma_1,\ldots,l_n:\sigma_n>:*}$$

Here $\mathcal{L}$ is the set of all labels.
The required domain equations are

$$Dom_{[\![\Gamma\vdash<l_1:\sigma_1,\ldots,l_n:\sigma_n>:*]\!]\eta}\ \cong\ \prod_{l_i\in\{l_1,\ldots,l_n\}} Dom_{[\![\Gamma\vdash\sigma_i:*]\!]\eta}$$

so we get

$$Dom_{[\![<l_1:\sigma_1,\ldots,l_n:\sigma_n>]\!]}\ \cong\ GP(<Dom_{[\![\sigma_i]\!]}\mid l_i\in\{l_1,\ldots,l_n\}>)$$

The subtyping rule for record-types is

$$\frac{\Gamma\vdash\sigma_1\leq\tau_1,\ldots,\sigma_m\leq\tau_m \quad m\leq n}{\Gamma\vdash<l_1:\sigma_1,\ldots,l_n:\sigma_n>\ \leq\ <l_1:\tau_1,\ldots,l_m:\tau_m>}(\leq REC)$$

and the associated coherence conditions are

$$\begin{array}{ll} & Coe_{[\![<l_1:\sigma_1,\ldots,l_n:\sigma_n>]\!][\![<l_1:\tau_1,\ldots,l_m:\tau_m>]\!]} \\ = & \\ & \Phi^{-1}_{<l_i:\tau_1,\ldots,l_m:\tau_m>} \\ & \circ GP(<\ Coe_{[\![\sigma_i]\!][\![\tau_i]\!]}\mid l_i\in\{l_1,\ldots,l_m\}>)\circ<proj_{l_i}\mid l_i\in\{l_1,\ldots,l_M\}> \\ & \circ\Phi_{<l_1:\sigma_1,\ldots,l_n:\sigma_n>}\end{array}$$

where

$$<proj_{l_i}\mid l_i\in\{l_1,\ldots,l_M\}>\ \in\ (\prod_{l_i\in\{l_1,\ldots,l_n\}} Dom_{[\![\sigma_i]\!]}) \longrightarrow (\prod_{l_i\in\{l_1,\ldots,l_m\}} Dom_{[\![\sigma_i]\!]})$$

When record types are added in this way, the models will also provide the semantics for record updates. It remains to be seen, which of the operations on records and record types mentioned in [CM89] can be modelled in this way.

Labelled sums, or variants, and bounded Σ-types can be treated in the same way as bounded Π-types and labelled products. Instead of the generalized product functor GP we use the generalized sum functor.
Another possible extension of the systems is to allow abstraction not only of terms over types but of terms over all kinds, and the corresponding form of application, i.e. terms to kinds. The system we then get is $F\omega$ ($\lambda\omega$ in Barendregt's cube [Bar9]) extended with subtyping and recursive types (but without recursion on higher kinds). To model the polymorphic types $\Pi(\Lambda\alpha : \kappa.\sigma)$ we also use the GP-functor, only this time applied to a family indexed by $Kind_\kappa$ instead of $Kind_*$.

Acknowledgements

I would like to thank Marc Bezem, Huub ten Eikelder and Kees Hemerik for helpful advice and discussions on this paper.

Appendix : Coherence

We will now prove that the semantics is coherent if the coherence conditions $\mathcal{P}_0$,$\mathcal{P}_1$, $\mathcal{P}_2$ and $\mathcal{P}_3$ hold. We use the fact that we have *minimal typing* in $\Lambda_{\leq}$:

36 lemma (minimal typing)
In a given context Γ every term M has a *minimal* type, i.e. a type σ_{min} such that

$$\Gamma \vdash M : \sigma_{min} \ \text{ and } \ \forall_\sigma\ \Gamma \vdash M : \sigma \ \Rightarrow\ \Gamma \vdash \sigma_{min} \ \leq\ \sigma$$

proof by induction on M
□

Type derivations for a term are for a large part determined by the syntax of that term. If we have a derivation for $\Gamma \vdash M : \sigma$, then the syntax of M determines which is the last rule other than *(SUB)* used in that derivation. For instance, if $\Gamma \vdash (\lambda x : \sigma.M) : \sigma \to \tau$ the last rule other than *(SUB)* used in the derivation must be $(\to I)$. We cannot tell by the syntax of a term if and where the rules *(SUB)* may have been used in a type derivation.

First a few words about notation.

- By $[\![\, \Gamma \vdash M : \sigma \,]\!]$ we mean the function $(\boldsymbol{\lambda}\eta\ .\ [\![\, \Gamma \vdash M : \sigma \,]\!]\eta)$ from environments η, $\eta \models \Gamma$, to $\bigcup_\eta Dom_{[\![\Gamma\vdash\sigma:*]\!]\eta}$.

- Suppose Δ is a derivation deriving $\Gamma \vdash M : \tau$ from $\Gamma_1 \vdash N_1 : \sigma_1 \ldots \Gamma_n \vdash N_n : \sigma_n$ i.e.

$$\frac{\dfrac{\Gamma_1 \vdash N_1 : \sigma_1}{\vdots} \quad \cdots \quad \dfrac{\Gamma_n \vdash N_n : \tau_n}{\vdots}}{\Gamma \vdash M : \tau}$$

  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 $\mathcal{R}_\Delta$ such that

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

- 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.

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

37 lemma For all derivations Δ :

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

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

proof follows directly from $\mathcal{P}_0$ and $\mathcal{P}_1$.

□

38 lemma For all type inference rules (T) not equal to *(SUB)* all derivations Δ,

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

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

proof

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

Suppose

$$\Delta\ :\ \frac{\Gamma \vdash M : \sigma_1 \rightarrow \sigma_2 \quad \Gamma \vdash N : \sigma_3}{\Gamma \vdash MN : \tau}(\rightarrow E)$$

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

$$\Delta\ :\ \frac{\dfrac{\dfrac{M : \sigma_1 \rightarrow \sigma_2}{M : \rho_1 \rightarrow \rho_2} \quad \dfrac{N : \sigma_3}{N : \rho_1}}{MN : \rho_2}(\rightarrow E)}{MN : \tau}$$

Using $\mathcal{P}_2$, we can prove that $\mathcal{R}_\Delta$ does not depend on ρ_1 and ρ_2.

$$\begin{aligned}
& \Phi_{\rho_1 \rightarrow \rho_2} [\![M : \rho_1 \rightarrow \rho_2]\!] \eta \\
= \ & \{ \text{ lemma 37} \} \\
& \Phi_{\rho_1 \rightarrow \rho_2}(Coe_{\sigma_1 \rightarrow \sigma_2\ \rho_1 \rightarrow \rho_2} [\![M : \sigma_1 \rightarrow \sigma_2]\!] \eta) \\
= \ & \{\mathcal{P}_2\} \\
& \Phi_{\rho_1 \rightarrow \rho_2}((\Phi^{-1}_{\rho_1 \rightarrow \rho_2} \circ FS(Coe_{\rho_1\ \sigma_1}, Coe_{\sigma_2\ \rho_2}) \circ \Phi_{\sigma_1 \rightarrow \sigma_2}) [\![M : \sigma_1 \rightarrow \sigma_2]\!] \eta) \\
= \ & \{ \text{ definition } FS\} \\
& \Phi_{\rho_1 \rightarrow \rho_2}(\Phi^{-1}_{\rho_1 \rightarrow \rho_2} \circ Coe_{\sigma_2\ \rho_2} \circ (\Phi_{\sigma_1 \rightarrow \sigma_2} [\![M : \sigma_1 \rightarrow \sigma_2]\!] \eta) \circ Coe_{\rho_1\ \sigma_1}) \\
= \ & \{\Phi_{\rho_1 \rightarrow \rho_2} \text{ is a bijection}\} \\
& Coe_{\sigma_2\ \rho_2} \circ (\Phi_{\sigma_1 \rightarrow \sigma_2} [\![M : \sigma_1 \rightarrow \sigma_2]\!] \eta) \circ Coe_{\rho_1\ \sigma_1}
\end{aligned}$$

and using this we can prove

$[\![\, MN : \tau \,]\!]\eta$

$=$ { lemma 37}

$Coe_{\rho_2\ \tau}[\![\, MN : \rho_2 \,]\!]\eta$

$=$ {definition $[\![\ \ ]\!]$ for $(\rightarrow E)$}

$Coe_{\rho_2\ \tau}((\Phi_{\rho_1\rightarrow\rho_2}[\![\, M : \rho_1 \rightarrow \rho_2 \,]\!]\eta)[\![\, N : \rho_1 \,]\!]\eta)$

$=$ { lemma 37}

$Coe_{\rho_2\ \tau}(\ (\Phi_{\rho_1\rightarrow\rho_2}[\![\, M : \rho_1 \rightarrow \rho_2 \,]\!]\eta)\ (\ Coe_{\sigma_3\ \rho_1}[\![\, N : \sigma_3 \,]\!]\eta)\ )$

$=$

$(\ Coe_{\rho_2\ \tau}\circ(\Phi_{\rho_1\rightarrow\rho_2}[\![\, M : \rho_1 \rightarrow \rho_2 \,]\!]\eta)\circ\ Coe_{\sigma_3\ \rho_1})[\![\, N : \sigma_3 \,]\!]\eta\ )$

$=$ { see above}

$(\ Coe_{\rho_2\ \tau}\circ\ Coe_{\sigma_2\ \rho_2}\circ(\Phi_{\sigma_1\rightarrow\sigma_2}[\![\, M : \sigma_1 \rightarrow \sigma_2 \,]\!]\eta)\circ\ Coe_{\rho_1\ \sigma_1}\circ\ Coe_{\sigma_3\ \rho_1})[\![\, N : \sigma_3 \,]\!]\eta$

$=$ $\{2\times\ \mathcal{P}_1\ \}$

$(\ Coe_{\sigma_2\ \tau}\circ(\Phi_{\sigma_1\rightarrow\sigma_2}[\![\, M : \sigma_1 \rightarrow \sigma_2 \,]\!]\eta)\circ\ Coe_{\sigma_3\ \sigma_1})[\![\, N : \sigma_3 \,]\!]\eta$

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

□

39 theorem (coherence)

All derivations of $\Gamma \vdash M : \tau$ give the same meaning $[\![\, \Gamma \vdash M : \tau \,]\!]\eta$.

proof by induction on M.

<u>base</u>

M is a variable or a constant : trivial.

<u>step</u>

Suppose we have two derivations, Δ_1 and Δ_2 , for $\Gamma \vdash M : \tau$. Then these derivations must end with the same rule, so they are of the following form

$$\Delta_1 : \quad \dfrac{\dfrac{\dfrac{\Delta_{11}}{\Gamma_i \vdash N_1 : \sigma_1} \quad \cdots \quad \dfrac{\Delta_{1n}}{\Gamma_n \vdash N_n : \sigma_n}}{\Gamma \vdash M : \sigma}(T)}{\Gamma \vdash M : \tau} \qquad \Delta_2 : \quad \dfrac{\dfrac{\dfrac{\Delta_{21}}{\Gamma_1 \vdash N_1 : \rho_1} \quad \cdots \quad \dfrac{\Delta_{2n}}{\Gamma_n \vdash N_n : \rho_n}}{\Gamma \vdash M : \rho}(T)}{\Gamma \vdash M : \tau}$$

By the induction hypothesis, all derivations for $\Gamma \vdash N_i : \sigma_i$ yield the same meaning $[\![\, \Gamma \vdash N_i : \sigma_i \,]\!]$, and the same is true for $\Gamma \vdash N_i : \rho_i$.

So in Δ_j each Δ_{ji} can be replaced by any derivation we want, and the resulting derivation will give the same meaning for $\Gamma \vdash M : \tau$ as Δ_j.

We will now use the fact that 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 :

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

But by lemma 38 for all derivations Δ

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

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

□

Using lemma 16 and the examples on page 27, we can actually show that the semantics is coherent if and only if $\mathcal{P}_0$,$\mathcal{P}_1$,$\mathcal{P}_2$ and $\mathcal{P}_3$ hold.

References

[ABL86] R. Amadio, K. B. Bruce, and G. Longo. The finitary projection model for second order lambda calculus and higher order domain equations. In *Logic in Computer Science*, pages 122–135. IEEE, 1986.

[Bar9] H. P. Barendregt. Typed lambda calculi. In D. M. Gabbai, S. Abramsky, and T. S.E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 1. Oxford University Press, 199-. 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.

[BL88] Kim B. Bruce and Giuseppe Longo. A modest model of records, inheritance and bounded quantification. In *Logic in Computer Science*, pages 38–50. IEEE, 1988.

[BMM90] Kim B. Bruce, Albert R. Meyer, and John C. Mitchell. The semantics of second-order lambda calculus. *Information and Computation*, 85:76–134, 1990.

[BTCGS89] V. Breazu-Tannen, Th. Coquand, C. A. Gunter, and A. Scedrov. Inheritance and explicit coercion. In *Logic in Computer Science*, pages 112–129. IEEE, 1989.

[CG90] Pierre-Louis Curien and Giorgio Ghelli. Cohenence of subsumption. In A. Arnold, editor, *CAAP*, pages 132–146. Springer LNCS, 1990.

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

[CM89] Luca Cardelli and John C. Mitchell. Operations on records. In M. Main et al, editor, *Fifth International Conference on Mathematical Foundations of Programming Semantics*, volume 442 of *LNCS*, pages 22–53, 1989.

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

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

[Gir72] J.-Y. Girard. *Interprétation fonctionelle et élimination des coupures de l'arithmétique d'ordre supé]rieur.* PhD thesis, Université Paris VII, 1972.

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

[HS73] Horst Herrlich and George E. Strecker. *Category Theory.* Allyn and Bacon, 1973.

[Mac79] N. MacCracken. *An Investigation of a Programming Language with a Polymorphic Type Structure.* PhD thesis, Syracuse University New York, 1979.

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

[MP88] John C. Mitchell and Gordon D. Plotkin. Abstract types have existential type. *ACM Trans. on Prog. Lang. and Syst.*, 10(3):470–502, 1988.

[Pol91] Erik Poll. Some category-theoretical properties for a model for second order lambda calculus with subtyping. Computing Science Note ??, Eindhoven University of Technology, 1991.

[Rey74] John C. Reynolds. Towards a theory of type structure. In *Programming Symposium: Colloque sur la Programmation*, LNCS, pages 408–425. Springer, 1974.

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

[tEH89a] H. ten Eikelder and C. Hemerik. The construction of a cpo model for second order lamba calculus with recursion. In *Procs. CSN'89 Computing Science in the Netherlands*, pages 131–148, 1989.

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

[tEM88] H. ten Eikelder and R. Mak. Language theory of a lambda calculus with recursive types. Computing Science Note 14, Eindhoven University of Technology, 1988.

In this series appeared:

89/1	E.Zs.Lepoeter-Molnar	Reconstruction of a 3-D surface from its normal vectors.
89/2	R.H. Mak P.Struik	A systolic design for dynamic programming.
89/3	H.M.M. Ten Eikelder C. Hemerik	Some category theoretical properties related to a model for a polymorphic lambda-calculus.
89/4	J.Zwiers W.P. de Roever	Compositionality and modularity in process specification and design: A trace-state based approach.
89/5	Wei Chen T.Verhoeff J.T.Udding	Networks of Communicating Processes and their (De-)Composition.
89/6	T.Verhoeff	Characterizations of Delay-Insensitive Communication Protocols.
89/7	P.Struik	A systematic design of a parallel program for Dirichlet convolution.
89/8	E.H.L.Aarts A.E.Eiben K.M. van Hee	A general theory of genetic algorithms.
89/9	K.M. van Hee P.M.P. Rambags	Discrete event systems: Dynamic versus static topology.
89/10	S.Ramesh	A new efficient implementation of CSP with output guards.
89/11	S.Ramesh	Algebraic specification and implementation of infinite processes.
89/12	A.T.M.Aerts K.M. van Hee	A concise formal framework for data modeling.
89/13	A.T.M.Aerts K.M. van Hee M.W.H. Hesen	A program generator for simulated annealing problems.
89/14	H.C.Haesen	ELDA, data manipulatie taal.
89/15	J.S.C.P. van der Woude	Optimal segmentations.
89/16	A.T.M.Aerts K.M. van Hee	Towards a framework for comparing data models.
89/17	M.J. van Diepen K.M. van Hee	A formal semantics for Z and the link between Z and the relational algebra.

90/1	W.P.de Roever- H.Barringer- C.Courcoubetis-D.Gabbay R.Gerth-B.Jonsson-A.Pnueli M.Reed-J.Sifakis-J.Vytopil P.Wolper	Formal methods and tools for the development of distributed and real time systems, p. 17.
90/2	K.M. van Hee P.M.P. Rambags	Dynamic process creation in high-level Petri nets, pp. 19.
90/3	R. Gerth	Foundations of Compositional Program Refinement - safety properties - , p. 38.
90/4	A. Peeters	Decomposition of delay-insensitive circuits, p. 25.
90/5	J.A. Brzozowski J.C. Ebergen	On the delay-sensitivity of gate networks, p. 23.
90/6	A.J.J.M. Marcelis	Typed inference systems : a reference document, p. 17.
90/7	A.J.J.M. Marcelis	A logic for one-pass, one-attributed grammars, p. 14.
90/8	M.B. Josephs	Receptive Process Theory, p. 16.
90/9	A.T.M. Aerts P.M.E. De Bra K.M. van Hee	Combining the functional and the relational model, p. 15.
90/10	M.J. van Diepen K.M. van Hee	A formal semantics for Z and the link between Z and the relational algebra, p. 30. (Revised version of CSNotes 89/17).
90/11	P. America F.S. de Boer	A proof system for process creation, p. 84.
90/12	P.America F.S. de Boer	A proof theory for a sequential version of POOL, p. 110.
90/13	K.R. Apt F.S. de Boer E.R. Olderog	Proving termination of Parallel Programs, p. 7.
90/14	F.S. de Boer	A proof system for the language POOL, p. 70.
90/15	F.S. de Boer	Compositionality in the temporal logic of concurrent systems, p. 17.
90/16	F.S. de Boer C. Palamidessi	A fully abstract model for concurrent logic languages, p. p. 23.
90/17	F.S. de Boer C. Palamidessi	On the asynchronous nature of communication in logic languages: a fully abstract model based on sequences, p. 29.

90/18	J.Coenen E.v.d.Sluis E.v.d.Velden	Design and implementation aspects of remote procedure calls, p. 15.
90/19	M.M. de Brouwer P.A.C. Verkoulen	Two Case Studies in ExSpect, p. 24.
90/20	M.Rem	The Nature of Delay-Insensitive Computing, p.18.
90/21	K.M. van Hee P.A.C. Verkoulen	Data, Process and Behaviour Modelling in an integrated specification framework, p. 37.
91/01	D. Alstein	Dynamic Reconfiguration in Distributed Hard Real-Time Systems, p. 14.
91/02	R.P. Nederpelt H.C.M. de Swart	Implication. A survey of the different logical analyses "if...,then...", p. 26.
91/03	J.P. Katoen L.A.M. Schoenmakers	Parallel Programs for the Recognition of *P*-invariant Segments, p. 16.
91/04	E. v.d. Sluis A.F. v.d. Stappen	Performance Analysis of VLSI Programs, p. 31.
91/05	D. de Reus	An Implementation Model for GOOD, p. 18.
91/06	K.M. van Hee	SPECIFICATIEMETHODEN, een overzicht, p. 20.
91/07	E.Poll	CPO-models for second order lambda calculus with recursive types and subtyping, p.
91/08	H. Schepers	Terminology and Paradigms for Fault Tolerance, p. 25.
91/09	W.M.P.v.d.Aalst	Interval Timed Petri Nets and their analysis, p.53.
91/10	R.C.Backhouse P.J. de Bruin P. Hoogendijk G. Malcolm E. Voermans J. v.d. Woude	POLYNOMIAL RELATORS, p. 52.
91/11	R.C. Backhouse P.J. de Bruin G.Malcolm E.Voermans J. van der Woude	Relational Catamorphism, p. 31.
91/12	E. van der Sluis	A parallel local search algorithm for the travelling salesman problem, p. 12.
91/13	F. Rietman	A note on Extensionality, p. 21.
91/14	P. Lemmens	The PDB Hypermedia Package. Why and how it was built, p. 63.

91/15	A.T.M. Aerts K.M. van Hee	Eldorado: Architecture of a Functional Database Management System, p. 19.
91/16	A.J.J.M. Marcelis	An example of proving attribute grammars correct: the representation of arithmetical expressions by DAGs, p. 25.
91/17	A.T.M. Aerts P.M.E. de Bra K.M. van Hee	Transforming Functional Database Schemes to Relational Representations, p. 21.
91/18	Rik van Geldrop	Transformational Query Solving, p. 35.
91/19	Erik Poll	Some categorical properties for a model for second order lambda calculus with subtyping, p. 21.
91/20	A.E. Eiben R.V. Schuwer	Knowledge Base Systems, a Formal Model, p. 21.
91/21	J. Coenen W.-P. de Roever J.Zwiers	Assertional Data Reification Proofs: Survey and Perspective, p. 18.
91/22	G. Wolf	Schedule Management: an Object Oriented Approach, p. 26.
91/23	K.M. van Hee L.J. Somers M. Voorhoeve	Z and high level Petri nets, p. 16.
91/24	A.T.M. Aerts D. de Reus	Formal semantics for BRM with examples, p. .
91/25	P. Zhou J. Hooman R. Kuiper	A compositional proof system for real-time systems based on explicit clock temporal logic: soundness and complete ness, p. 52.
91/26	P. de Bra G.J. Houben J. Paredaens	The GOOD based hypertext reference model, p. 12.
91/27	F. de Boer C. Palamidessi	Embedding as a tool for language comparison: On the CSP hierarchy, p. 17.
91/28	F. de Boer	A compositional proof system for dynamic proces creation, p. 24.