

$\beta\eta$ -complete models for system F .

Stefano Berardi & Chantal Berline.

June 7th, 1998

Abstract

We show that the model of system F introduced in [5], and having as polymorphic maps exactly all Scott continuous maps, is $\beta\eta$ -complete. This proves in particular the existence of non trivial $\beta\eta$ -complete models for F .

1 Introduction.

In this paper we will progress in the study of non-trivial models of the notion of polymorphic maps of Girard's System F [16] and Reynolds polymorphism [25]. This study was started in Scott [28],[29], Girard [17], and continued by Coquand-Gunter-Winskel [12]. Our contribution is to prove that there exist non trivial $\beta\eta$ -complete models of F .

More precisely we will prove that the model introduced in [5], with this precise purpose in mind, is $\beta\eta$ -complete. This model, called here the BB -model for short, is a Scott model, and the polymorphic maps in it are exactly the Scott continuous ones ; furthermore its definition is very concrete and has no direct link with the syntax of F .

We will not need here the precise definition of the BB -model. Rather, we will isolate a set of conditions and we will prove that they ensure completeness. As proved in [5], the BB -model satisfies all these conditions, hence it is $\beta\eta$ -complete.

The BB -model is generalized in [7] to a whole class of feasible models of F , in which, in particular, it is very easy to check for most of the conditions, whether they are satisfied or not. This class contains a lot of other complete models.

By contrast, none of the previously known extensional models of System F satisfy all conditions and most are known to be incomplete ; this is discussed in the next section and in section 3.3.

In order to have a reasonably general frame where to express our conditions we will introduce a simple-minded definition of models of System F , which covers all the models in [7], and a few other classes. This abstract class of models is also

convenient since it allows a very simple interpretation of F -terms. It however only covers models with “the maximum number of polymorphic maps” and hence rules out all the models of F which have constrained sets of polymorphic maps. This does not matter here since all known such (non syntactic) models are indeed incomplete.

We introduce now the plan of the paper.

In section 2 we sketch the problematic about polymorphic maps, about their models, and then the problem of $\beta\eta$ -completeness. We in particular explain why, in our opinion, achieving $\beta\eta$ -completeness is a step towards the study of polymorphic maps. Eventually we list the questions we leave open.

In section 3 we recall the definition of system F and give the restricted definition of models of system F which is suitable for our purpose ; then we outline the method used in the completeness proof. The interpretation of system F w.r.t. such models is made explicit in section 7.

In section 4 we introduce the polymorphic maps we will use in the proof.

In sections 5, 6 we prove the completeness result.

We can add that we make a detailed presentation of the syntax of F to fix the terminology and recall the constraints, because we use them deeply in the completeness proof. We also made a detailed presentation of the interpretations of terms in our models, since it is not really standard, even if it looks familiar, and since the existing presentations, in general much more complex (for example [3] or [11]), were not directly usable.

2 Problematics about polymorphic maps and completeness.

In this section we suppose the reader already acquainted with simply typed λ -calculus, syntactic and semantic models, and to know the bare definition of system F , which is recalled anyhow later on.

Polymorphic maps and the problematic about them. Call type constructor any function $\Phi : Types \rightarrow Types$, where $Types$ is the class of types, and types are sets. Then polymorphic maps are maps associated to some type constructor Φ , taking as input any type α , and returning as output some element of $\Phi(\alpha)$. A trivial example is the polymorphic identity id , associated to the type constructor $Id(\alpha) = (\alpha \rightarrow \alpha)$. The polymorphic map id takes as input α , and returns the identity $id(\alpha)$ of type $(\alpha \rightarrow \alpha)$.

The type of the polymorphic maps associated to Φ is denoted $\forall\alpha.\Phi(\alpha)$. System F (see Section 3) is the smallest extension of simply typed lambda calculus closed under such operations over types. The naive approach is to say that, in a model of system F , $\forall\alpha.\Phi(\alpha)$ is a subset the set-theoretical cartesian product $\prod\alpha \in Types.\Phi(\alpha)$, since this set consists exactly of all maps f sending any $\alpha \in Types$ to some $f(\alpha) \in \Phi(\alpha)$ (provided $\Phi(\alpha)$ is non empty). However the most natural attempt fails : it is indeed impossible to expand the standard

model of simply typed λ -calculus (which interprets $\alpha \rightarrow \beta$ as the full set of functions from α to β) into a model of F , as shows Reynolds [26]. Therefore, for interpreting F (via functional models) one has necessarily to take smaller interpretations of the arrow.

We will now describe what is already known concretely about this problem, that is, the (more or less concrete) models of polymorphism we already have. We will justify the choice of a particular model to study. Eventually, we will explain our contribution to the topic.

Models of polymorphic maps. In realizability or PER models of F , which originate in Girard and Troelstra [16],[31], polymorphic types are all realized by constant maps. Thus, a PER model, while it is a useful tool for studying the properties of F as a programming language, it is too restrictive for studying polymorphism in full generality. This last sentence also apply to “parametric” models, which require that polymorphic maps behave as uniformly as possible (see [21] for a survey). It should hence be clear that we will not be concerned here with this kind of models.

The first models with non-constant polymorphic maps, that we will call here *universal retraction models* or *u.r.* models, were introduced, for the continuous semantics, by D. Scott and McCracken in [28], [29], [20] and continued by Amadio-Bruce-Longo [2]. In these models terms are interpreted as elements of a model of the untyped λ -calculus, and types are ranges of a (suitable subclass of) retractions of the model, and identified with such retractions. The word “universal” refers to the fact that in these models there is a type of all types. In *u.r.* models the polymorphic maps associated to Φ consist exactly of all continuous maps f sending any $\alpha \in Types$ to some $f(\alpha) \in \Phi(\alpha)$. Then Berardi [6] showed that similar work could be done for the stable semantics, taking this time the whole class of stable retractions (see [9] for a survey). The only limit of these constructions were the complexity in the definition of the set *Types*. Such complexity has forbidden, up now, any extensive study of their properties.

Then a categorical model was introduced by Girard in [17]. Girard interpreted *Types* as the category of coherent domains and embeddings, and type constructors as functors “continuous” and “stable” (in a categorical sense) over *Types*. The model was universal in the sense that second order quantification was over all coherent domains, and not only subdomains of a given domain. Moreover the model was economical and it interpreted polymorphic maps f associated to a type constructor Φ as all families of elements $f(X) \in \Phi(X)$, indexed over $X \in Types$, and satisfying the following requirement:

for each embedding $j : X \rightarrow Y$, $f(X)$ is the “trace of $f(Y)$ over $\Phi(j)$ ” (in a sense he made precise).

As a consequence, the set of all polymorphic maps associated to some Φ , with the stable ordering, was a coherent space, chosen as the interpretation of $\forall \alpha. \Phi(\alpha)$ in *Types*. Again, the only limit of this model construction, if we are interested in studying the properties of polymorphic maps, was its complexity.

Then Coquand-Gunter-Winskel [12] started some (at first, only conceptual) simplification of Girard’s construction. They showed that it could be carried

over the category of Scott domains and embedding, dropping all the “stability” requirements. They interpreted type constructors as “continuous” functors (again, in a categorical sense).

By building over the two preceding works, Barbanera-Berardi [5] showed that, if we restrict the embeddings of the category *Types* to the canonical inclusion maps, we further simplify this construction. In this paper we will study some relevant properties of this model, whose interest is that it combines strong properties with simplicity of construction.

In the Barbanera-Berardi model (BB-model) *Types*, the category of Scott domains, is replaced by a single Scott domain *Types*, whose elements are themselves Scott domains. *Types* is ordered by the restriction order (between the information systems underlying Scott domains). The union *Terms* of all types is still a Scott domain, interpreting the terms of F . Type constructors are now exactly the Scott-continuous functions Φ over *Types*. The polymorphic maps associated to such a Φ are now interpreted as all Scott-continuous maps $f : \text{Types} \rightarrow \text{Terms}$, such that $f(X) \in \Phi(X)$ for all $X \in \text{Types}$. Here no category theory is really needed and *Types* is completely understood.

We claim also that the interpretation of types and terms in the BB-model, and more generally in all models in [7], is simpler than in the *u.r.* models (when comparison is possible, see [7]). Like *u.r.* models, these models take care of typed terms as such (they do not forget types).

The BB-model might be further generalized taking κ -continuous maps¹, for κ any regular cardinal; it is not clear how far we might go in adding polymorphic maps to the model.

A reason to think that the BB-model is already a quite general model of polymorphism is that it is the first non-trivial model known to be $\beta\eta$ -complete.

To prove this is our contribution to the topic.

We will now explain what $\beta\eta$ -completeness is, and why we think that it is a relevant property of models.

$\beta\eta$ -Completeness. A model is $\beta\eta$ -complete for a typed lambda calculus (just “complete”, for short) if and only if it equates only $\beta\eta$ -convertible term. The trivial examples of $\beta\eta$ -complete model are the $\beta\eta$ term models. The term model of System F was the only complete model of F known up to now.

Negative examples of completeness.

First there are obvious sources of incompleteness linked to the lack polymorphic maps in a model (this is related to the possible “parametricity” of the model, which is often considered as desirable, from a programming point of view). The simplest of them is the following : suppose that the interpretation of a type σ is a singleton, then the model equates x^σ and y^σ ; this is the case for example in the models of Girard and CGW. Consider now FC , the equational theory studied by Longo, Milsted and Soloviev in [19], namely the equational

¹The κ -Scott topology is a weakening of Scott topology (case $\kappa = \omega$), which admits more and more monotonous functions as κ increases. The ω_1 -case was introduced by Plotkin [24] for modelling parallelism (see [13]) and greater cardinals are used for foundational purposes ([14], [10]).

theory of F plus “Axiom C ”. This axiom is indeed a scheme, which requires that $t^{\forall\alpha.\sigma}\tau = t^{\forall\alpha.\sigma}\tau'$ for all types σ, τ, τ' such that α is not free in σ , and all term t of the relevant type. It is clear that a model where all the polymorphic maps associated to a constant constructor Φ are themselves constants, will satisfy axiom C . Since Axiom C is independent of F and $F\eta$, such a model can't be complete w.r.t. these two equational theories. This is the case for the *PER* models, and for Girard's model for example.

We now give another source of incompleteness, which is also linked to a lack of polymorphic maps. Let N be the type of integers in F (defined in [17]). Suppose that the interpretation of N in a *PER* model contains only the interpretations of closed terms of type N (this is for example the case with the *PER* models built over the term model of F or over P_ω). Then the model equates two closed F -terms $f, g : N \rightarrow N$ if and only if $f(n) =_\beta g(n)$ for all closed $n : N$ (see [8] for a proof). Such a model equates for example left and right addition, which have however an obviously different computational meaning.

A positive example of completeness.

Friedman [15] showed that all full models of simply typed lambda calculus which include integers in the base type are $\beta\eta$ -complete (where “full” means that $\sigma \rightarrow \tau$ is interpreted as the set of *all* set-theoretical maps from σ to τ).

Apparently, a model may distinguish two non- $\beta\eta$ -convertible terms by applying them to “very generic” maps. For instance, term models distinguish between non-convertible terms by applying them to variables, and getting different results. And variables are in fact a (trivial) example of “very generic” maps. In Friedman's proof, full models distinguish between non-convertible terms by applying them to “very generic” maps, built by the Choice Axiom of Set Theory.

Another possibility for having completeness was discovered by Simpson [27], by building over a syntactic result of Statman [30]. Simpson proved that all models including integers in the base type, and sum and product over such integers, are complete. In this case, the completeness uses as starting point a strong property of sum and product: using them, one can encode the entire *term* model of simply typed lambda calculus inside just the base type of the model itself.

Contribution of the paper w.r.t. polymorphism. The previous discussion suggests that if we find a complete model of F , then it should contain polymorphic maps of arbitrary shape, or at least a few ones able to encode the term model.

Indeed, when proving that BB-model is complete, we also discovered that it contains some polymorphic maps able to produce such encoding (see section 4). Such curious maps are not, for instance, in any *PER* model; in particular we use polymorphic case functions testing over the shapes of types. In this way, this paper contributed to the study of polymorphism, showing how (polymorphic) discrimination over types (Cond.II.4-5) permits the maximum discrimination over terms, provided that *Types* is rich enough, nicely structured, and that we dispose of some standard computation tools.

Related remarks and open questions on completeness.

- We treat $\beta\eta$ -completeness here since our proof relies on some kinds of “logical relations, which, as usual logical relations, do not allow to distinguish elements with the same applicative behavior. The question of finding a β -complete model for F is hence still open, and we propose in [7] a simple candidate.
- Simpson [27] proved completeness for all models of simply typed λ -calculus including integers, sum and product, by proving they are complete for terms of type T (where T is the type of binary trees). Then he used a result of Statman [30] for deducing completeness at *every* type. We do not know if this may happen for System F , that is, if there is a type σ of F such that a model complete for terms of type σ is complete for all terms (but we know that this is false for T).
- Also, it is an open problem to know whether there are β - or $\beta\eta$ - complete models of *untyped* lambda calculus in the continuous or stable semantics. A significant step in this direction is [13], where Di Gianantonio- Honsell-Plotkin show that there is a $\beta\eta$ -complete model in a variant of the ω_1 -continuous semantics. However the model in [13] is built by an inverse limit construction starting from the term model itself (viewed as a flat domain) ; the situation is different with the BB-model, whose construction does not refer at all to the term model. The completeness result in this paper might hence support the conjecture that more natural $\beta\eta$ -models might exist.
- We do not pretend of course that the set of conditions for completeness we give in Section 3.3 is optimal, and we hope that a more elegant, and maybe more general, set of conditions will be provided by further workers.

3 System F and its models.

In this section we briefly introduce system F (for more details we refer to [17]). Then we define models of system F , using a definition we found more suitable for the completeness proof than those from the literature (for example [11] or [3]). This definition requires no more category theory than knowing what is a c.c.c. (even the notion of functor is of no explicit use here) and is abstract enough to fit different settings (for example all *r.u.* models, the class in [7], and all models of untyped λ -calculus when viewed as models of F).

The definition is not general enough to give an account of Girard’s and CGW’s models, of the PER models, and not even of the stable analogue of the class in [7], which will be treated in a future paper (the reasons are given later on). It is more than likely that this definition could be weakened to include (some) models with constrained sets of polymorphic maps, however we noticed that, even generalizing it for covering our stable class, which would be the simplest of them, already raised technical difficulties. Since there was no need

here to include models which are obviously incomplete, we decided to stick to this simple definition.

By contrast, our definition covers non extensional models, even if the bare completeness proof here only works for F_η . The reason is that the result can then be used to prove that some models of [7] have a theory included in F_η , and in particular allows to isolate good candidates for being complete for F . This choice is not at all costly: roughly speaking, one just finds here retraction pairs where one would find isomorphisms in the extensional case.

3.1 System F .

Definition of system F runs as follows.

A *type*, or *F-type*, is either a type variable, an arrow type or a polymorphic type. Types are constructed using the following schemes :

- (type) variables α, α_i, \dots are types ($i \in N$).
- $\sigma \rightarrow \tau$ is a type iff σ and τ are,
- $\forall\alpha.\sigma$ is a type iff α is a type variable and σ is a type ; the variable α is bounded in $\forall\alpha.\sigma$.

Except when we will encode the syntax, types will be considered only up to α -equivalence (renaming of bounded variables) and $\sigma[\alpha : \tau]$ will then denote the type obtained by correct substitution of the type τ to all free occurrences of the variable α within the type σ .

A *term*, or *F-term*, is either a variable, an abstraction, an application, a type abstraction or a type application. Terms come equipped with a built-in type. Terms and their types are constructed using the following schemes :

- (term) variables $x^\sigma, y^\sigma, \dots$, where σ can be any type, are terms of type σ .
- $\lambda x^\sigma.t$ is a term of type $\sigma \rightarrow \tau$ iff t is a term of type τ ; the variable x^σ is bounded in $\lambda x^\sigma.t$.
- tu is a term of type τ iff t is a term of type $\sigma \rightarrow \tau$ and u is a term of type τ .
- $\lambda\alpha.t$ is a term of type $\forall\alpha.\sigma$ iff t is a term of type σ and α is not free in the type of any free term-variable of t ; the variable α is bounded in $\lambda\alpha.t$.
- $t\sigma$ is a term of type $\tau[\alpha : \sigma]$ iff t is a term of type $\forall\alpha.\tau$.

We will write $t : \sigma$ or t^σ to mean that t is an F -term of type σ . The rules are such that the type is uniquely determined from the F -term.

Except when we will encode the syntax, terms will be considered only up to α -equivalence, and $t[x^\sigma : u]$ will then denote the term obtained by correct substitution of the term u of type σ to all free occurrences of x^σ within the term

t . We also need $t[\alpha : \sigma]$; this is defined only if α is not free in any free term variable of t , and then means the obvious thing.

Immediate β -reduction is defined by :

$(\lambda x^\sigma . t)u^\sigma \rightarrow t[x^\sigma : u^\sigma]$ and

$(\lambda \alpha . t)\sigma \rightarrow t[\alpha : \sigma]$ (recall α cannot be free in any free term variable of t).

Immediate η -reduction is defined by :

$\lambda x^\sigma . tx^\sigma \rightarrow t$ if $t : \sigma \rightarrow \tau$ for some τ and x^σ is not free in t .

$\lambda \alpha . t\alpha \rightarrow t$ if $t : \forall \alpha' . \sigma$ for some α', σ and α is not free in t .

Immediate $\beta\eta$ -reduction as the union of both.

Then we define β - (*resp.* $\beta\eta$ -) *reduction* as the smallest transitive relation containing the corresponding immediate reduction, and compatible with the formation of terms. Both reductions are Church-Rosser. Finally we define β - (*resp.* $\beta\eta$ -) *equivalence* (or *convertibility*) as the reflexive, symmetric and transitive closure of the corresponding reduction; these relations are denoted by $=_\beta$ and $=_{\beta\eta}$.

3.2 Models of F .

By a model for system F we will mean the following tuple,

$$\mathcal{M} = \langle Univ, \langle Types, Terms \rangle, \langle \Rightarrow, lbd, apl \rangle, \langle Q, Lambda, Appl \rangle \rangle$$

where, $Univ$ is a category, which will be the ambient universe where lives our model, $Types$ and $Terms$ are two objects of $Univ$, $Q, \Rightarrow, Lambda$ and $Appl$ are four morphisms of $Univ$, and finally lbd and apl are two families of morphisms of $Univ$. Besides the necessary precisions on the domain and codomain of the morphisms, and on the indexing of the families, this tuple is subject to the requirements listed below, together with the motivations behind. The first requirement is that $Univ$ is a pleasant *c.c.c.*, two of the requirements assert the existence of convenient retraction pairs, and the other ones are simple compatibility conditions.

Recall that a *retraction pair* between two objects A, B of a category is, by definition, a pair (f, g) of morphisms such that $g \circ f = id_A$. We then say that A is a *retract* of B and that f is *left invertible*.

1. **We assume** that $Univ$ is a cartesian closed category with enough points [4, p.108]. Thus it is concrete and we may think that objects of $Univ$ are real sets (plus additional structure), and morphisms are real functions.

By definition of cartesian closed, $Univ$ is equipped with a cartesian product \times , and contains, for each pair of objects (A, B) : an object $A \rightarrow B$ representing $Hom(A, B)$. For sake of readability we assume here that $Hom(A, B)$ is an object of $Univ$, which is taken as $A \rightarrow B$, and that *eval* is the usual application of a function to its argument ².

²Thus, in the present setting, $A \rightarrow B$ and $Hom(A, B)$ denote exactly the same mathematical object. We keep the two notations here since: first \rightarrow is more convenient for readability,

2. *Types* will contain the interpretation of all F -types. By analogy with syntax the elements of *Types*, denoted by X, Y, Z , will be called *semantic types*, and $x \in X$ will also be written $x : X$.

We require $Types \neq \emptyset$ and that the elements of *Types* are non empty objects of *Univ*.

3. *Terms*, as well as $\cup Types$, will contain the interpretation of all F -terms.

We require that $\cup Types \subseteq Terms$.

4. $\Rightarrow \in Hom(Types \times Types, Types)$ is a morphism and it will be the interpretation of the arrow constructor on types.

5. *lbd* and *apl* are indexed by $Types \times Types$.

We require that $X \rightarrow Y$ is a retract of $X \Rightarrow Y$, for all $X, Y \in Types$, and that $(lbd_{X,Y}, apl_{X,Y})$ is a retraction pair between both.

The family (lbd, apl) will interpret abstraction over a term variable, and application of a term to a term.

6. $Q \in Hom((Types \rightarrow Types), Types)$.

Second order quantification will be interpreted by Q , and $Q(F)$ will also be denoted $\forall X.F(X)$ for $F \in Types \rightarrow Types$.

7. **We require** that $Types \rightarrow Terms$ is a retract of *Terms*, and that $(Lambda, Appl)$ is a retraction pair between both.

Lambda and *Appl* will interpret abstraction of terms over type variables and application of terms to types.

8. For all “type constructor” $F \in Hom(Types, Types)$ we let now $Hom_F(Types, Terms)$ be the subset of $Hom(Types, Terms)$ containing those morphisms f such that $f(X) \in F(X)$ for all $X \in Types$.

Our last requirement is that *Lambda* and *Appl* induce functions $Lambda_F$ and $Appl_F$ between $Hom_F(Types, Terms)$ and $Q(F)$ ³:

if $f \in Hom_F(Types, Terms)$, then $Lambda(f) \in Q(F) \subseteq Terms$ and,

if $u \in Q(F)$ and $X \in Types$, then $Appl(u)(X) \in F(X)$.

Out of any model \mathcal{M} for F , and for any assignment ρ of elements of *Types* to type variables of F , and of elements of *Terms* to term variables of F , in a

while *Hom* allows to stress that one deals with morphisms and functions, and second since the generalisation to the more general setting would require to write things down this way (and moreover to add non trivial maps *eval* and Λ).

On the other hand, the distinction between \rightarrow and the \Rightarrow which will interpret implication needs to be done: as it is pointed out later on, in concrete examples $X \rightarrow Y$ and $X \Rightarrow Y$ can only be, in the best cases, isomorphic.

³If *Univ* is some standard ccc (Scott’s, Berry’s, a.s.o.) conditions 7+8 imply that $Hom_F(Types, Terms)$ is a retract of $Q(F)$. But in general $Hom_F(Types, Terms)$ need not even be an object of *Univ*.

compatible way, we may define an interpretation $[\cdot]_\rho$ of F -types and F -terms. This is detailed in Section 7 (Appendix).

A model \mathcal{M} of F identifies $(\alpha)\beta$ -convertible terms and α -convertible types. It identifies η -convertible terms, and is then labelled *extensional*, if we ask that (lbd, apl) is a family of inverse isomorphisms, and that $(\Lambda_F, \text{Appl}_F)$ is a pair of inverse bijections.

Example 1 1. “Trivial models”. Let (M, lbd, apl) be a model of untyped λ -calculus in a ccc $Univ$ (here lbd and apl are morphisms and form a retraction pair), we let $Types := \{M\}$, $Terms := M$. We have $(M \Rightarrow M) = M$ and $Q(F) = M$ for the only possible F . Hence $Q(F)$ is always homomorphic to $\text{Hom}_F(Types, Terms)$ while $M \Rightarrow M$ is isomorphic to $M \rightarrow M$ iff M is extensional as a model of untyped λ -calculus. This example clearly shows the links between \Rightarrow and \rightarrow . However it is easily seen that trivial models can’t be complete, since they satisfy Axiom C because all polymorphic maps are constant.

2. In the BB-model, $X \Rightarrow Y$ is the cpo of traces of elements of $X \rightarrow Y$, for all $X, Y \in Types$, and hence both cpos are isomorphic. More generally in the models of [7] $X \Rightarrow Y$ “contains” a copy of the cpo of traces, and $X \rightarrow Y$ can also be here a proper retract of $X \Rightarrow Y$. The situation is similar for Q .

3. In all r.u. models $X \Rightarrow Y$ is the range of a retraction, and is directly isomorphic to $X \rightarrow Y$ (with no intermediate encoding via traces), and similarly with Q .

Remark 2 We insist that $X \rightarrow Y$, and more generally $A \rightarrow B$, is never a semantic type in our setting, while $X \Rightarrow Y$ is.

Remark 3 The two last conditions, which imply in particular that $\text{Hom}_F(Types, Terms)$ is “smaller” than $Q(F)$ is the very condition which rules out all the classes of models that we mentioned in the introduction of this section, since it means that all “possible” polymorphic functions will be in $\forall X. F(X)$. We do not know yet how to weaken this without increasing technicity significantly.

Remark 4 In case $\text{Hom}_F(Types, Terms)$ is an object of $Univ$, which is indeed the case for all our concrete models, then we could give an alternative formulation to 7-8, requiring that: “for all $F \in Types \rightarrow Types$ there is a retraction pair $(\Lambda_F, \text{Appl}_F)$ between $\text{Hom}_F(Types, Terms)$ and $Q(F)$. This would not increase technicity, but would give less readable interpretations of polymorphic terms.

Remark 5 The condition that $\emptyset \notin Types$ is of course satisfied in the four classes which fit exactly the definition above (continuous or stable r.u. models, continuous models of [7], trivial models), but it is also satisfied in Girard’s and CGW models. It is convenient because, then, the interpretation of a type σ is never empty, and term-environments are everywhere defined functions (which can take the value \perp). This would not be the case in the PER models. When this condition is not satisfied, then one has to complicate the definition of “the model satisfies $t^\sigma = u^\sigma$ ”, while here it is just saying that both are equated in every environment.

Notational conventions for applications. All conceivable notions of applications will be denoted along the pattern $\psi(\xi)$. This precisely means that we adopt the following abbreviations :

$$\begin{array}{llll}
f(x) \text{ is usual and} & \in Y & \text{if } f \in \text{Hom}(X, Y) = X \rightarrow Y, & x \in X \\
f(x) := \text{appl}_{X, Y}(f)(x) & \in Y & \text{if } f \in X \Rightarrow Y, & x \in X \\
f(X) \text{ is usual and} & \in \text{Terms} & \text{if } f \in \text{Hom}(\text{Types}, \text{Terms}), & X \in \text{Types} \\
f(X) := \text{Appl}(f)(X) & \in \text{Terms} & \text{if } f \in \text{Terms}, & X \in \text{Types} \\
F(X) \text{ is usual and} & \in \text{Types} & \text{if } F \in \text{Hom}(\text{Types}, \text{Types}), & X \in \text{Types}
\end{array}$$

We will also need the application $F(X)$ of a type to a type. The reason is that we will have to quantify over F in some cases, for example to define $\forall F.F(X)$ and, then, F has to range over types, not over elements of $\text{Types} \rightarrow \text{Types}$. For this we suppose fixed a morphism $AP \in \text{Hom}(\text{Types}, \text{Types} \rightarrow \text{Types})$ which allows to translate F to an element of $\text{Types} \rightarrow \text{Types}$, in order to give a meaning to $F(X)$. Indeed, one of our sufficient conditions for completeness is the existence of a left inverse AP for Q , thus we have just anticipated a little. Let us make now our definitions of application (of a type to a type) and of second order quantification $\forall X$ precise :

$$\begin{array}{llll}
F(X) := AP(F)(X) & \in \text{Types} & \text{if } F \in \text{Types}, & X \in \text{Types} \\
\forall X.F(X) := Q(F) & \in \text{Types} & \text{if } F \in \text{Hom}(\text{Types}, \text{Types}) & \text{as already defined} \\
\forall X.F(X) := Q(AP(F)) & \in \text{Types} & \text{if } F \in \text{Types} &
\end{array}$$

Notation 6 We turn now to “semantic abstractions”:

$$\begin{array}{ll}
\lambda x^X.f(x^X) \text{ means } \text{lbd}_{X, Y}(f) & \text{if } f \in \text{Hom}(X, Y) \\
\lambda X.f(X) \text{ means } \text{Lambda}(f) & \text{if } f \in \text{Hom}(\text{Types}, \text{Terms})
\end{array}$$

This notation is compatible with the preceding one in the sense that :

$$\begin{array}{l}
(\lambda x^X.f(x^X))(u) = f(u) \text{ for all } u \in X \\
(\lambda X.f(X))(Z) = f(Z) \text{ for all } Z \in \text{Types}
\end{array}$$

Notation 7 As usual $A, B \rightarrow C$ abbreviates $A \rightarrow (B \rightarrow C)$, and similarly for \Rightarrow , and $\forall X.A \Rightarrow B$ abbreviates $\forall X.(A \Rightarrow B)$. We will use Curryfication freely.

We will sometimes use $x : X$ for $x \in X \in \text{Types}$.

Finally we will write : $(f, g) : A \rightleftarrows B$ to mean that (f, g) is a pair of morphisms $f \in \text{Hom}(A, B)$ and $g \in \text{Hom}(B, A)$.

3.3 The completeness result.

Rather than proving directly completeness of BB-model, we now isolate a group of conditions on models, which are true for BB-model and are sufficient to imply completeness. These conditions apply to many models in [7].

We first describe the conditions and then, in the rest of the section, we sketch the proof that they imply completeness. The rest of the paper will contain the real proof.

Fix any model \mathcal{M} of F as in the previous subsection. We divide the conditions on \mathcal{M} into two groups.

- I. The first group of conditions requires that we have infinitely many infinite types, and that we may develop partial recursion inside \mathcal{M} , through a conditional and fixed point operators. They are easy to satisfy, and, indeed, true for most models. We express them as follows:
1. We have an object $O \in Types$ having an infinite subset N whose elements are representative of all integers.
 2. There is a 1-1 morphism $\langle \perp, \perp \rangle \in Hom(O \times O \rightarrow O)$ with left inverse (p_1, p_2) , and such that if $x, y \in N$, then $\langle x, y \rangle \in N$ and $\langle x, y \rangle$ is strictly bigger than x, y .
 3. There is a uniform fixed point operator $Y \in \forall X.(X \Rightarrow X) \Rightarrow X$, such that $Y(X)$ is a fixed point operator over X : for all $X \in Types$ and $f \in X \Rightarrow X$ we have $f(Y(X)(f)) = Y(X)(f)$.
 4. There is a fixed point operator $Y_K \in (K \rightarrow K) \rightarrow K$ for any $K \in Univ$. There is also a conditional operator $IF_K : O, O, K, K \rightarrow K$, such that for all $k, k' \in N$,
 $IF_K(k, k', a, b) = a$ if $k = k'$ and b otherwise.

- II. The second group of conditions is the crucial one for the completeness proof, and is not satisfied by models which are “poor” in polymorphic functions. It mainly says that the basic constructions over types (the arrow operator \Rightarrow , the quantification operator Q) are left invertible. A consequence will be that we may reconstruct the syntactic form of a type σ out of the interpretation $[\sigma]_\rho$ of σ in $Types$. Since the assignment of a value to a type variable is not reversible in general we need to specify that there are canonical values for type variables, an infinite family of semantic types $C(i)$, and that C is a left invertible morphism.

For technical reasons we need a further (minor) condition which says that we may compute the “trace” of any type over any other one. Making all this precise now, we require that :

1. There is a retraction pair $(C, index) : O \rightleftarrows Types$. If i is any integer, then we will think of $C(i) \in Types$ as “the canonical value for the type variable α_i ”. All $C(i)$'s have to be infinite. We express this by saying that O is, uniformly, a retract of all $C(i)$'s. That is, there are two families $f \in \forall X.[O \Rightarrow C(index(X))]$, $g \in \forall X.[C(index(X)) \Rightarrow O]$ such that $f(C(i)), g(C(i))$ is an embedding-retraction pair between O and $C(index(C(i))) = C(i)$. We will also use C_i for $C(i)$.
2. There are morphisms $P_1, P_2 \in Hom(Types, Types)$ reversing \Rightarrow . In other words, for all $X, Y \in Types$, we have :
 $P_1(X \Rightarrow Y) = X$, $P_2(X \Rightarrow Y) = Y$.

3. There is a morphism $AP \in \text{Hom}(\text{Types}, \text{Types} \rightarrow \text{Types})$ reversing Q . In other words : for all “type constructors” $F \in \text{Types} \rightarrow \text{Types}$ we have $AP(Q(F)) = F$.
4. We have a case map $\text{case} \in \forall G. \forall X. G(X), G(X), G(X) \Rightarrow G(X)$ distinguishing among arrow types, quantified types, and “canonical values for variables”. By this we mean that for all $G \in \text{Types}$, and $a, b, c \in \forall X. G(X)$ we have :

$$\begin{aligned} \text{case}(G)(X)(a(X), b(X), C(X)) &= a(X) \text{ if } X := C(i). \\ \text{case}(G)(X)(a(X), b(X), C(X)) &= b(X) \text{ if } X := Y \Rightarrow Z \\ \text{case}(G)(X)(a(X), b(X), C(X)) &= c(X) \text{ if } X := Q(F). \end{aligned}$$
5. There is a “trace operator” between any two types, that is, some “map” $j \in \forall X. (\forall Y. X \Rightarrow Y)$, computing the “trace of $x \in X$ in a type Y ”. The only requirement we make over such j is that $j(X, X, x) = x$ for all $X \in \text{Types}$. We make no assumption over the value of $j(X, Y, x)$ in any other case ; in a domain-theoretical model of F , this value will often be \perp (an element representing an “undefined” result).

Example 8 *Some of these conditions are easy to satisfy :*

1. *All the models of [7] satisfy Conditions II.3 and II.5, and many of them satisfy Condition II.2.*
2. *Trivial models obviously satisfy II.2,3,5, and do not satisfy II.1,4.*
3. *All r.u. models satisfy II.5 and do not satisfy II.2 (and we do not know whether r.u. models can be complete).*
4. *PER models do not satisfy II.3 and II.4 and are not complete, since they satisfy FC.*
5. *Girard’s model does not satisfy II.3 (Q is not even injective ⁴), does not satisfy II.4 since it is a model of FC, and does not satisfy II.5 since the corresponding type is trivial there. We slightly conjecture that it satisfies II.2, but, there, \Rightarrow is a functor, and we have to check what happens on morphisms, which is not immediate (by contrast it is immediate to check whether a model of [7] satisfies II.2 or not).*
6. *The situation with the continuous CGW is similar for II.2, II.3, and II.5, but the argument given for II.4 doesn’t apply since the continuous CGW is not a model of FC.*

Theorem 9 *(Completeness Theorem)*

1. *The BB-model is extensional and satisfies the two groups of conditions.*
2. *Every model \mathcal{M} satisfying the two groups of conditions is $\beta\eta$ complete, that is:*

for all terms t, u of F , if $t, u : A$ then $t =_{\beta\eta} u$ if and only if for all assignment $\rho, \mathcal{M} \models \rho(t) = \rho(u)$.

⁴The analogue of our Q is not injective in Girard’s model since, for example, both $\forall \alpha. \alpha$ and $\forall \alpha. \forall \beta. \alpha \rightarrow \beta$ take value $\{\emptyset\}$ in it (see [17] for the first type).

For a proof of point 1 we refer to [5]. In the rest of the section we will sketch the proof of point 2, proof that we will develop from the next section on.

We want to adapt Friedman completeness proof for simply typed lambda calculus to system F (we failed, up to now, to adapt to F the much simpler and stronger Statman-Simpson argument).

The idea is to choose an infinite type in \mathcal{M} , and encode the $\beta\eta$ term model of System F in it. Then we will prove that there is a 1-1 correspondence between the interpretations of terms in (such encoding of) the term model and in \mathcal{M} . From this we will deduce that if two terms are equated in \mathcal{M} , then they are equated in the $\beta\eta$ -term model, hence they are $\beta\eta$ -convertible.

We chose to encode the $\beta\eta$ term-model of system F in the type O of \mathcal{M} : both terms and types will be encoded by integers. The construction (included in section 4) would fail in most models : the second group of conditions we assumed is essential.

We need the fact that we may reconstruct, through a polymorphic map $type : \forall X.O$, the (integer code of the) syntax of a type σ from its interpretation $[\sigma]_\rho$ in \mathcal{M} . Such $type([\sigma]_\rho)$ will, indeed, return (the integer code in O of) σ , provided ρ is a canonical assignment (provided $\rho(\alpha_i) = C(i)$). The existence of $type$ will be derived from the two groups of conditions we assumed. Then, in section 5, we will adapt Plotkin's method of logical relations, and use it in section 6 to prove that there is a 1-1 correspondence between the interpretations of terms in the term model and in \mathcal{M} ; $\beta\eta$ -completeness will follow.

We end this section with a side remark.

Remark 10 *One may ask why we restricted the statement of $\beta\eta$ -completeness to the case where t, u have the same type. The reason is that we consider meaningful an equation $t = u$ only when t, u have the same type. In fact, a model may equate terms of different type: for instance, $\lambda x^N . x^N$ and $\lambda \alpha . \lambda x^\alpha . x^\alpha$ are equated in any PER model. We do not know whether this curious and unpleasant phenomenon may happen in the BB-model, but we conjecture it does not. We conjecture more generally that this cannot happen in any of the models in [7].*

4 Some new polymorphic maps.

In this section we deduce, from the assumption that the model \mathcal{M} satisfies our two groups of conditions, the existence of some other polymorphic maps that we will use in the proof that \mathcal{M} is $\beta\eta$ -complete.

4.1 Encodings.

In this subsection we will prove the existence, in \mathcal{M} , of integer maps coding types and terms of F , and then prove that we may reconstruct a term from its applicative behavior on such encodings.

4.1.1 Types and terms encodings.

For sake of simplicity, we keep the same notation for an integer and its representation in O .

We now fix an explicit encoding of types and pseudoterms in O (*pseudoterms* are built like terms, but without any compatibility restrictions). Both encodings ($\#_{tp}$ for types, and $\#_{tr}$ for terms) are simply denoted by $\#$ when it is clear from the context which one we are using.

We chose as coding of types:

$$\begin{aligned} \#\alpha_i &:= \text{var}(i) &:= \langle 0, i \rangle \\ \#(\sigma \rightarrow \tau) &:= \text{arrow}(\#\sigma, \#\tau) &:= \langle 1, \langle \#\sigma, \#\tau \rangle \rangle \\ \#\forall\alpha_i.\tau &:= \text{forall}(\#\alpha_i, \#\tau) &:= \langle 2, \langle \#\alpha_i, \#\tau \rangle \rangle \end{aligned}$$

We choose as coding of terms:

$$\begin{aligned} \#x_i^\sigma &:= \text{Var}(i, \#\sigma) &:= \langle 3, \langle i, \#\sigma \rangle \rangle \\ \#(tu) &:= \text{Ap1}(\#t, \#u) &:= \langle 4, \langle \#t, \#u \rangle \rangle \\ \#t\tau &:= \text{Ap2}(\#t, \#\tau) &:= \langle 5, \langle \#t, \#\tau \rangle \rangle \\ \#\lambda x_i^\sigma.t &:= \text{Lb1}(\#x_i^\sigma, \#t) &:= \langle 6, \langle \#x_i^\sigma, \#t \rangle \rangle \\ \#\lambda\alpha_i.t &:= \text{Lb2}(\#\alpha_i, \#t) &:= \langle 7, \langle \#\alpha_i, \#t \rangle \rangle \end{aligned}$$

Both encodings are injective maps, respectively between the set of types or pseudo-terms into $N \subseteq O$.

Notation 11 We will denote by \check{O}_{tp} the set of all codes of F -types. We will denote by \check{O} the subset of N which is the set of all codes of $F \perp$ terms, by \check{O}_σ the set of codes of F -terms of type σ , and by \check{O}_\forall the union of all those \check{O}_σ such that σ begins with \forall . Because of the typing rules of F , two α -equivalent σ, σ' give the same \check{O}_σ .

4.1.2 Transferring the basic equivalences to O .

We also introduce two equivalence relations over O .

- $x \sim_{tp} y$ if x and y encode two α -equivalent types or $x \equiv y$, and
 - $x \sim_{tr} y$ if x and y encode two $\alpha\beta\eta$ -equivalent well-typed terms or $x \equiv y$.
- Once more we use a generic name “ \sim ” when more convenient.

These equivalence relations are external to the model (and not recursive). They will however allow us to see the term model within \mathcal{M} .

A first immediate remark is that the morphisms *arrow*, *Ap1* and *Ap2* are compatible with \sim_{tp} and \sim_{tr} . Otherwise stated, all these morphisms induce operations on O / \sim . More accurately, if we turn now to the family $T_\sigma := \check{O}_\sigma / \sim_{tr}$ then some suitable restrictions of the morphisms above induce operations between the T_σ 's. Such operations allow to view $(T_\sigma)_\sigma$ as the term model of F_η . This will be partly made explicit in the following section.

4.1.3 Reconstructions.

We show here that we can reconstruct (the code of) a term, up to equivalence, from its extensional behavior (in the sense of its action on all possible variables).

We will in fact define some higher- order functions reconstructing a term. That these functions are morphisms will follow from the fact that they are obtained from the polytime morphisms and the basic morphisms of the underlying *c.c.c.* (composition, evaluation, a.s.o.).

Lemma 12 *Let e be a type (resp. a term), let ξ and ξ_i be term or type variables of the same kind and such that the following makes sense, finally let $f : O \rightarrow O$. Then :*

$$\forall i \in N (f(i) \sim \#(e[\xi : \xi_i])) \implies e \simeq e_f[\xi_r : \xi]$$

where $r := f(0) + 1$ and $\#e_f := f(r)$, and \simeq denotes α - or $\alpha\beta\eta$ - equivalence.

Proof. Let e'_f be such that $\#e'_f = f(0)$. By assumption, $e'_f \simeq e[\xi : \xi_0]$. By Condition I.2, $\langle a, b \rangle > a, b$, therefore we have $\#\xi_r \geq r = f(0) + 1 > \#e'_f$. By applying I.2 several times, we deduce that ξ_r cannot appear in e'_f , otherwise it would follow $\#\xi_r < \#e'_f$. Thus, either ξ_r is not free in $e[\xi : \xi_0]$ or (in the term-cases) there is a $\beta\eta$ -reduct of $e[\xi : \xi_0]$ in which ξ_r is not free (just take a common reduct e'' of $e[\xi : \xi_0]$ and e'_f ; then, ξ_r is not free in e'' since ξ_r is not free in e'_f). This is equivalent to saying that either $\xi = \xi_r$ or ξ_r is not free in some reduct of e . It is easy to see that, in both cases, we have $e \simeq e[\xi : \xi_r][\xi_r : \xi]$. Now, by definition of e_f we have $e_f \simeq e[\xi : \xi_r]$. We conclude $e \simeq e_f[\xi_r : \xi]$.

Corollary 13 1. (Reconstruction of terms). *There is a morphism $La1 : O, (O \rightarrow O) \rightarrow O$ such that, for all type σ and all $g : O \rightarrow O$, and all F -term t ,*

$$\text{if } g(\#x_i^\sigma) \sim_{tr} t[x^\sigma : x_i^\sigma] \text{ for all } i \in N,$$

$$\text{then } La1(\#g) \sim_{tr} \#(\lambda x^\sigma. t)$$

2. (Reconstruction of polymorphic terms). *There is a morphism $La2 : (O \rightarrow O) \rightarrow O$ such that, for all $h : O \rightarrow O$, all F -term t , and all α which is not free in the type of a free variable of t ,*

$$\text{if } h(\#\alpha_i) \sim_{tr} t[\alpha : \alpha_i] \text{ for all } i \in N,$$

$$\text{then } La2(h) \sim_{tr} \#(\lambda \alpha. t)$$

3. (Reconstruction of types). *There is a morphism $quant : (O \rightarrow O) \rightarrow O$ such that, for all $k : O \rightarrow O$, and all type τ ,*

$$\text{if } k(i) \sim_{tp} \#(\tau[\alpha : \alpha_i]) \text{ for all } i \in N$$

$$\text{then } quant(k) \sim_{tp} \#(\forall \alpha. \tau).$$

Proof. We only treat the second case, the other two ones are similar. Applying the preceding lemma to $f := h \circ \text{var}$, $\xi := \alpha$, and $\xi_i = \alpha_i$, we get $t \simeq t_f[\alpha_r : \alpha]$ and α_r is not free in the type of a free variable of t_f , hence $\lambda\alpha.t \simeq \lambda\alpha_r.t_f$. So :

$$\begin{aligned} \#(\lambda\alpha.t) &\sim_{tr} \#(\lambda\alpha_r.t_f) = \text{Lb2}(\text{var}(r), \#t_f) ; \text{ thus :} \\ \#(\lambda\alpha.t) &\sim_{tr} \text{La2}(h) \text{ with } \text{La2}(h) := \text{Lb2}(\text{var}(h(0) + 1), h(h(0) + 1)). \end{aligned}$$

Remark 14 (*Extensionality of La1, La2, and quant*). *If the values of g, g' and h, h' are \sim_{tr} -equivalent on all codes of variables, and the values of k, k' are \sim_{tp} -equivalent on all $i \in N$, then $\text{La1}(\#\sigma, g) \sim_{tr} \text{La1}(\#\sigma, g')$, $\text{La2}(h) \sim_{tr} \text{La2}(h')$ and $\text{quant}(k) \sim_{tp} \text{quant}(k')$.*

Remark 15 • $\text{La1}(\#\sigma, \text{Ap1}(v)) \sim_{tr} v$ for all type σ and all $v \in \check{O}_{\sigma \rightarrow \tau}$.

Follows from the definition of La1, applied to $g := \text{Ap1}(v) := \Lambda w. \text{Ap1}(v, w)$.

Let indeed $v := \#t$, then $g(\#x_i^\sigma) = \#(tx_i^\sigma)$ (by definition of Ap1) = $\#((ty^\sigma)[y^\sigma : x_i^\sigma])$, where y^σ is a fresh term-variable, hence $\text{La1}(\#\sigma, \text{Ap1}(v)) \sim_{tr} \#(\lambda y^\sigma. ty^\sigma) \sim_{tr} \#t = v$.

• $\text{La2}(\text{Ap2}(v)) \sim_{tr} v$ for all $v \in \check{O}_\forall$.

The proof is similar and follows from the definition of La2 applied to $h := \text{Ap2}(v) := \Lambda w. \text{Ap2}(v, w)$.

4.2 The polymorphic map *type* reconstructing a type.

In this subsection we will define a map *type* computing the (integer code of) a type out of its interpretation in \mathcal{M} . This map will be crucial in order to get a coding of the term model of F inside \mathcal{M} , coding we will use in order to prove that \mathcal{M} is $\beta\eta$ -complete.

4.2.1 Definition by case and by recursion.

The first stage in the definition of *type* is to derive, out of the two groups of conditions we assumed, definitional schemata for polymorphic maps in \mathcal{M} .

Lemma 16 (*Definition by cases*). *let $F : \text{Types} \rightarrow \text{Types}$ and $a, b, c : \forall X. F(X)$.*

Then there is some $f : \forall X. F(X)$ such that :

1. $f(X) = a(X) : F(X)$ if $X = C_i$.
2. $f(X) = b(X) : F(X)$ if $X = Y \Rightarrow Z$ for some Y, Z ,
3. $f(X) = c(X) : F(X)$ if $X = Q(K)$ for some $K : \text{Types} \rightarrow \text{Types}$.

Proof. Trivial, using the function *Case* given by Condition II.4: just take $f := \lambda X. \text{case}(Q(F))(X)(a(X), b(X), c(X))$

Lemma 17 (*Definition by recursion*). *Let $F : \text{Types} \rightarrow \text{Types}$ and let :*

- $$\begin{aligned} a &: \forall Z. F(C(\text{index}(Z))), \\ b &: \forall Y. \forall Z. (F(Y), F(Z) \Rightarrow F(Y \Rightarrow Z)), \\ c &: \forall G. [\forall Z. F(G(Z))] \Rightarrow F(\forall Z. G(Z)). \end{aligned}$$

Then there is some $f : \forall X.F(X)$ such that :

$$\begin{aligned} f(C_i) &= a(C_i) && : F(C_i) , \text{ for all } i \in N \\ f(Y \Rightarrow Z) &= b(Y)(Z)(f(Y), f(Z)) && : F(Y \Rightarrow Z) , \text{ for all } Y, Z : \text{Types} \\ f(Q(K)) &= c(Q(K))(\lambda Z.f(K(Z))) && : F(Q(K)) , \text{ for all } K : \text{Types} \rightarrow \text{Types} \end{aligned}$$

It has to be stressed here that the hypothesis on c implies that for all $K : \text{Types} \rightarrow \text{Types}$ we have : $c(Q(K)) : Q(F \circ K) \Rightarrow F(Q(K))$, that is to say :

$$c(Q(K)) : \forall Z.F(K(Z)) \Rightarrow F(\forall Z.K(Z)) (**).$$

In the hypothesis, we had to quantify over $G : \text{Types}$ rather than on K because generalization over $\text{Types} \rightarrow \text{Types}$ is not possible in our frame. We could simulate the morphisms K by the type G because, by Condition II.3, every K is $AP(G)$ for some $G : \text{Types}$.

Proof.

We may translate the thesis by an equation $f = H(f) : \forall X.F(X)$, equation that we will solve afterwards by $f := Y'(H)$, where $Y' := Y(\forall X.F(X))$ is the fixed point on the type $\forall X.F(X)$. Define $H(f) : \forall X.F(X)$ as follows. Put

$$\begin{aligned} a'(X) &:= a(X) && : F(C(\text{index}(X))) \\ b'(X) &:= b(P_1(X), P_2(X))(f(P_1(X)), f(P_2(X))) && : F(P_1(X) \Rightarrow P_2(X)) \\ c'(X) &:= c(X)(\lambda Z.f(X(Z))) && : F(\forall Z.X(Z)). \end{aligned}$$

Now we use the case definition lemma in order to define the body $H' := H(f)$ of H . We will also use the embedding j to assign to a', b', c' the type required by the definition by cases.

$$\begin{aligned} H'(X) &= j(F(C(\text{index}(X))), F(X), a'(X)) && : F(X) \quad \text{if } X := C_i \text{ for some } i \in N \\ H'(X) &= j(F(P_1(X) \Rightarrow P_2(X)), F(X), b'(X)) && : F(X) \quad \text{if } X := Y \Rightarrow Z \text{ for some } Y, Z \\ H'(X) &= j(F(Q(AP(X))), F(X), c'(X)) && : F(X) \quad \text{if } X := Q(K) \text{ for some } K \end{aligned}$$

Now, $C(\text{index}(X)) = C_i$ if $X := C_i$ for some i ,

$P_1(X) \Rightarrow P_2(X) = X$ if $X := Y \Rightarrow Z$ for some Y, Z , and, finally,

$Q(AP(X)) = X =$ if $X := Q(K)$ for some K .

Using three times the equation $j(Z, Z, z) = z$, we conclude:

$$\begin{aligned} H'(C_i) &= a(C_i) && : F(C_i) \\ H'(Y \Rightarrow Z) &= b(Y, Z)(f(Y), f(Z)) && : F(Y \Rightarrow Z) \\ H'(Q(K)) &= c(Q(K))(\lambda Z.f(K(Z))) && : F(Q(K)) \end{aligned}$$

Let now $H := \lambda f.H'$ and $\phi := Y'(H)$. Then $\phi = H(\phi)$. Therefore ϕ has all required properties. \diamond

This general definition by recursion will be used to build the crucial morphism *type* below, and also to build a uniform family of pairs of morphisms $(f_X, g_X) : O \rightleftharpoons X$, where X ranges over all semantic types (Section 5.3 below).

4.2.2 The pair $(\text{type}, [\perp])$.

The definition of *type* uses the general recursion principle developed in the previous subsection while the definition of $[\perp]$ uses a somewhat different recursion principle based on Condition I.4.

Definition 18 *The recursive definition of $\text{type} \in \forall X.O$ is :*

- $type(C_i) = var(i) = \#\alpha_i$
- $type(Y \Rightarrow Z) = arrow(type(Y), type(Z))$
- $type(Q(K)) = quant(i \in O \mapsto type(K(C_i)))$.

Proof. Apply the previous lemma to : $a := \lambda Z.var(index(Z))$, $b := \lambda Y.\lambda Z.arrow$, $c := \lambda G.\lambda g^{\forall Z.O} quant(i \in O \mapsto g(C(i)))$.

Definition 19 Let ρ be any semantic environment on types, and let π be a syntactical environment on types (that is : π substitutes types to type variables, cf. Appendix). We will say that π is compatible with ρ if we have, for all i :

$$\#(\pi(\alpha_i)) \sim_{tp} type(\rho(\alpha_i)).$$

If π is compatible with ρ then $\pi(\alpha_i)$ is determined, up to α -equivalence, by $\rho(\alpha_i)$. It is also useful to note that :

Remark 20 If π is compatible with ρ then, for all i , $\pi[\alpha : \alpha_i]$ is compatible with $\rho[\alpha : C_i]$; this is immediate since the value of the environments is unchanged on $\alpha_j \neq \alpha$, and since $\rho[\alpha : \alpha_i](\alpha) = C_i$ and $\pi[\alpha : \alpha_i](\alpha) = \alpha_i$ (and $type(C_i) = \#\alpha_i$)

In the sequel we keep the same notation for the environments and their extensions to a map on types.

The following example is a key one.

Example 21 Let ρ_C be the interpretation of types such that $\rho_C(\alpha_i) := C_i$. Then it is immediate from the definition of type that id (the identity on types) is compatible with ρ_C .

Lemma 22 If π is compatible with ρ then, for all σ , we have :

$$\#(\pi(\sigma)) \sim_{tp} type(\rho(\sigma)).$$

Proof.

We prove the Lemma by induction on σ (and for all compatible π, ρ at each step). The case where σ is a variable coincides with the hypothesis.

Case 1. Suppose $\sigma := \theta \rightarrow \tau$. Then :

$$\begin{aligned} type(\rho(\sigma)) &= type(\rho(\theta \rightarrow \tau)) = type(\rho(\theta) \Rightarrow \rho(\tau)) \\ &= arrow(type(\rho(\theta)), type(\rho(\tau))) \text{ (definition of } type) \\ &\sim_{tp} arrow(\#(\pi(\theta)), \#(\pi(\tau))) \text{ (induction hypothesis plus commutativity of } \\ &\text{arrow with } \sim) \end{aligned}$$

$$= \#(\pi(\theta) \rightarrow \pi(\tau)) \text{ (definition of } arrow)$$

$$= \#(\pi(\theta \rightarrow \tau)) = \#\pi(\sigma) \text{ (inductive definition of } \pi).$$

Case 2. Suppose $\sigma := \forall \alpha.\tau$. Then

$$\begin{aligned} type(\rho(\sigma)) &= type(\rho(\forall \alpha.\tau)) = type(Q(Z \mapsto \rho[\alpha : Z](\tau))) \\ &= quant(i \in O \mapsto type(\rho[\alpha : C_i](\tau))) \text{ (definition of } type) \end{aligned}$$

$$= quant(k) \text{ where } k \text{ is the morphism defined by :}$$

$$k(i) = type(\rho[\alpha : C_i](\tau)). \text{ Now,}$$

$$k(i) \sim_{tp} \#(\pi[\alpha : \alpha_i](\tau)) \text{ (induction hypothesis plus Remark 20).}$$

Using now the definition of *quant* and that of $\pi(\forall\alpha.\tau)$ (which is recalled in the Appendix on syntactical substitutions) we have :

$$\mathit{quant}(k) \sim_{tp} \#(\forall\alpha.\pi[\alpha : \alpha](\tau)) = \#(\pi(\forall\alpha.\tau)) \text{ (definition of } \mathit{quant}).$$

Thus $\mathit{type}(\rho(\forall\alpha.\tau)) \sim_{tp} \#(\pi(\forall\alpha.\tau))$ Q.E.D.

Corollary 23 *For all $x \in \check{O}_{tp}$ we have : $\mathit{type}(\rho_C(\#^{-1}(x))) \sim_{tp} x$.*

There is no reason a priori why the function $\rho_C \circ \#^{-1} : O \rightarrow \mathit{Type}$ (which, by the way, is not yet defined outside \check{O}), should be a morphism ; the following lemma shows however that this is essentially true.

Lemma 24 *There is a morphism $[\perp] : O \rightarrow \mathit{Types}$ such that, for all $x \in \check{O}_{tp}$ we have $\mathit{type}([x]) \sim_{tp} x$.*

Proof. We show a more general result, using, namely that there is a morphism

$[\perp]_- : O, (O \rightarrow \mathit{Types}) \rightarrow \mathit{Types}$ such that for any $r : O \rightarrow \mathit{Types}$ and for all type σ we have

$$[\#\sigma]_r = \rho_r(\sigma) \text{ where } \rho_r \text{ is the environment defined by } \rho_r(\alpha_i) := r(i) \quad (*)$$

Then we set $[\perp] := [\perp]_C$. That $[\perp]$ satisfies the desired properties follows now from the previous Corollary since, if we restrict x to \check{O}_{tp} , we have $[x] = \rho_C(\#^{-1}(x))$, by (*).

The definition of $[\perp]_-$ is by recursion and by cases (I.4), using invertibility of $\langle \cdot, \cdot \rangle$ (2). We leave it as an exercise, knowing that one has first to define $\mathit{subst} : (O \rightarrow \mathit{Types}), O, \mathit{Types}, O \rightarrow \mathit{Types}$ such that, for all $i, j \in N$, $\mathit{subst}(r, i, X, j) = r(i)$ if $j \neq i$ and X otherwise.

5 Logical Relations

5.1 The core idea for the proof of completeness.

There is a last step in adapting Friedman's completeness argument to a model \mathcal{M} of F . We have to show that there is a 1-1 correspondence between the interpretations of any term in (the coding in O of) the term model, and in \mathcal{M} itself. This will imply that if two terms have the same interpretation in \mathcal{M} , then they have the same interpretation in the term model, hence they are $\beta\eta$ -convertible.

Friedman defined such 1-1 correspondence by induction over the type of the term model. We cannot repeat this induction reasoning for system F . Indeed, in order to define the interpretation of a type $\forall\alpha.\sigma$ in \mathcal{M} , we need to know the interpretations of all types $\sigma[\alpha := \tau]$ in \mathcal{M} ; but some of them have a degree higher than the degree of $\forall\alpha.\sigma$.

To solve this we will combine Girard's candidates method together with Plotkin's method of using logical relations [22]. More precisely, we will define

O -relations and (f, g) -relations (some kinds of logical relations⁵ well-fitted for system F), which should be considered as "candidates" for the 1-1 correspondence we have to define.

Definition 25 *An O -relation is a binary relation $R \subseteq \check{O} \times X$, where X is any semantic type, such that $R(x, y)$ and $R(x', y)$ implies $x \sim_{tr} x'$.*

The existence of a 1-1 correspondence will be expressed by the following key result, to be proved in the next section, from which the completeness of the model follows immediately :

Proposition 26 *There is an environment ρ (which interprets types and terms in the model) and a family S^σ of O -relations such that :*

1. $S^\sigma \subseteq \check{O} \times \rho(\sigma)$ for all σ , and
2. for any closed F -term $t : \sigma$, we have $S^\sigma(\#t, \rho(t))$.

This ρ is, in fact, nothing else than the ρ_C defined in Example 1 above.

As usual, for proving such a result we cannot stay within closed terms but have to prove an "open" version by induction on t . The statement of the open version (Proposition 41 below) is not immediate here and we need some preliminary work.

Another crucial point is that S^σ has to be built by induction on σ and that the class of O -relations is too crude for this ; thus we have to replace it by an accurate subclass : that of "candidates".

5.2 Pseudo-retraction pairs and (f, g) -relations.

Definition 27 *Given a pair $(f, g) : O \rightrightarrows X$ and given $D \subseteq \check{O}$ which is closed under \sim_{tr} and non empty, we call (f, g) -relation between D and X , any $R \subseteq D \times X$ such that :*

1. $x \in D$ implies $R(x, f(x))$.
 2. $R(x, y)$ implies $x \sim_{tr} g(y)$.
 3. $x \sim_{tr} x'$ and $R(x, y)$ implies $R(x', y)$.
- D will be called the domain of the relation, and X its codomain.

By the second condition, any (f, g) -relation is an O -relation.

Definition 28 *Given $\emptyset \neq D \subseteq \check{O}$ closed under \sim_{tr} , a pseudo retraction pair (f, g) between D and X is a pair of morphisms $(f, g) : O \rightrightarrows X$ such that, for all $x \in D$, we have $g(f(x)) \sim_{tr} x$.*

⁵Logical relations have already been used several times for system F , in general for ruling functions out of a given model (see for ex. [1] and [21]). Contrarily to the definition in [1] the use of the word "logical relations" is usurped here, since their definition in the $\forall\alpha$. case is "non- logical" in that it depends on several arbitrary choices and encodings.

Of course, any pseudo retraction pair between D and X is also a pseudo retraction pair between D' and X for any $\emptyset \neq D' \subseteq D$ which is also closed under \sim_{tr} . Finally we say that (f, g) is a *pseudo retraction pair* if it is a pseudo retraction pair between some D, X .

Remark 29 *If R is an (f, g) -relation between D and X , then (f, g) is a pseudo-retraction pair between D and X .*

Conversely, given a pseudo retraction pair (f, g) between D and X there is a minimum (f, g) -relation R between D and X , which is defined by : $R(x, y)$ iff $x \in D$ and $\exists x' (x' \sim_{tr} x$ and $y = f(x'))$.

Remark 30 *Let us here motivate the use of general (f, g) -relations instead of the minimum ones (which would amount to work directly with the pseudo-retraction pairs (f, g)). We observe that, given an (f, g) -relation R , the set $R(x) := \{y / R(x, y)\}$, which only depends of the \sim_{tr} -equivalence class of x , lies between $f(\{x' / x' \sim_{tr} x\})$ and $g^{-1}(x)$. This gives us a crucial flexibility when handling the (f, g) -relations, while working only with the minimum ones would stick us to the lower bound and would prevent desired closure properties of the set of relations.*

5.3 Building a uniform (f_X, g_X) - family.

We make the construction in four steps sketched below, two of them being left to the reader:

Preliminary step : we describe how to obtain a pair $(f_{X \Rightarrow Y}, g_{X \Rightarrow Y}) : O \rightrightarrows (X \Rightarrow Y)$ out of two pairs $(f_X, g_X) : O \rightrightarrows X$ and $(f_Y, g_Y) : O \rightrightarrows Y$. Similarly we show how to obtain $(f_{Q(K)}, g_{Q(K)})$ out all the $(f_{K(X)}, g_{K(X)})$.

First step. The intention is to build (f_X, g_X) for any X , using recursion (Lemma 17). The problem is that we need polymorphic f, g 's and that $\forall X. O \rightarrow X$ makes no sense. Thus, we will rather use recursion to build a pair (f', g') , such that, for all X in *Types*, $f' \in \forall X. O \Rightarrow X$, and $g' \in \forall X. X \Rightarrow O$, the intended behavior of f', g' being dictated by the preliminary step. For sake of readability, $f'(X)$ and $g'(X)$ are denoted f'_X and g'_X .

Second step. We are now in position to extract three polymorphic maps playing the role of the a, b, c in Lemma 17. This will be left to the reader.

Third step. We let now $f_X := lbd_{O, X}(f'_X)$ and $g_X := lbd_{O, X}(g'_X)$, this family behaves in the way described in the first step, once more this will be left to the reader.

Of course, for general X , we have little information on f_X, g_X , and in many models, they will often return the value \perp . We will however show that this pair is a pseudo retraction pair whenever X is the interpretation of some F -type (lemma 37).

5.3.1 Preliminary step : intuitions on (f, g) .

First we note that from $(f, g) : A \rightrightarrows B$ and $(f', g') : A' \rightrightarrows B'$ we can define a pair

$(f'', g'') := (f, g) \rightrightarrows (f', g') : (A \rightarrow A') \rightleftarrows (B \rightarrow B')$, by :

$f''(h) := f' \circ h \circ g$ and $g''(h') := g' \circ h' \circ f$.

Second we can compose pairs in the standard way :

$(f', g') \circ (f, g) := (f' \circ f, g \circ g')$.

This makes it possible to lift any $(f_X, g_X) : O \rightleftarrows X$ and $(f_Y, g_Y) : O \rightleftarrows Y$ to a pair $(f_{X \Rightarrow Y}, g_{X \Rightarrow Y}) : O \rightleftarrows (X \Rightarrow Y)$ as follows : we compose the pair $(Ap1, La1(type(X))) : O \rightleftarrows (O \rightarrow O)$ with the pair $(f_X, g_X) \rightrightarrows (f_Y, g_Y) : (O \rightarrow O) \rightleftarrows (X \rightarrow Y)$, and then with the retraction pair $(lbd_{X,Y}, apl_{X,Y}) :$

$(f_{X \Rightarrow Y}, g_{X \Rightarrow Y}) := (lbd_{X,Y}, apl_{X,Y}) \circ ((f_X, g_X) \rightrightarrows (f_Y, g_Y)) \circ (Ap1, La1(type(X)))$

the explicit definition of $(f_{X \Rightarrow Y}, g_{X \Rightarrow Y})$ from (f_X, g_X) and (f_Y, g_Y) is then:

$$f_{X \Rightarrow Y} : = x \in O \mapsto lbd_{X,Y}(f_Y \circ Ap1(x) \circ g_X) \quad (1)$$

$$g_{X \Rightarrow Y} : = y \in X \Rightarrow Y \mapsto La1(type(X), g_Y \circ apl_{X,Y}(y) \circ f_X) \quad (2)$$

Defining pairs $: O \rightleftarrows Q(K)$ from all the pairs $O \rightleftarrows K(X)$, for $K \in Types \rightarrow Types$ needs the morphism $[\perp]$, which was defined in Section 4.2.2.

$$f_{Q(K)} : = x \in O \mapsto Lambda(X \in Types \mapsto f_{K(X)}(Ap2(x, type(X)))) \quad (3)$$

$$g_{Q(K)} : = y \in Q(K) \mapsto La2(r \in O \mapsto g_{K([r])}(Appl(y)([r]))) \quad (4)$$

5.3.2 First step : conditions on (f', g') .

From this preliminary step it follows that (f', g') should satisfy the following conditions, which are now written using the abbreviations introduced at the end of Section 3.2 (otherwise *Lambda*, *Appl*, and a lot of instances of *lbd*, *apl* would appear.

$$f'_{X \Rightarrow Y} : = \lambda x^O. \lambda z^X. f'_Y(Ap1(x)(g'_X(z))) \quad (5)$$

$$g'_{X \Rightarrow Y} : = \lambda y^{X \Rightarrow Y}. La1[type(X), r \in O \mapsto g'_Y(y(f'_X(r)))] \quad (6)$$

and

$$f'_{Q(K)} : = \lambda x^O. \lambda X. f'_{K(X)}(Ap2(x, type(X))) \quad (7)$$

$$g'_{Q(K)} : = \lambda y^{Q(K)}. La2[r \in O \mapsto g'_{K([r])}(y([r]))] \quad (8)$$

We also intend that the value of f', g' on the $C(i)$'s be given by Cond. II.1.

5.3.3 Second step : the recursive definition of (f', g') .

The existence of $f' : \forall X.(O \Rightarrow X)$ and $g' : \forall X.(X \Rightarrow O)$ satisfying the conditions above follows from the recursion Lemma 17 applied to $F(X) := (O \Rightarrow X) \times (X \Rightarrow O)$. Here $A \times B := \forall Z.(A, B \Rightarrow Z) \Rightarrow Z$ is the interpretation in the model of the product defined in System F .

We have then to exhibit three polymorphic maps a, b, c satisfying the hypothesis of the lemma. Map a is given by *Cond* II.1 and we let the reader convince himself that he could extract b from 5,6 and c from 7,8.

5.3.4 Third step.

We let the reader check that $f_X := lbd_{O,X}(f'_X)$ and $g_X := lbd_{O,X}(g'_X)$ satisfy the conditions stated in the preliminary step.

5.3.5 Typographical simplification.

From now on, (f_X, g_X) always refer to the family defined above. For sake of readability we will not mention the pairs (lbd, apl) and $(\text{Lambda}, \text{Appl})$ anymore, and will use “ \circ ” to denote all possible compositions in the model.

In particular there will be no visual distinction between the pair (f, g) and (f', g') and we will simply write :

- R.1.** $f(C_i)$ and $g(C_i)$ are given by *Cond*.II.1
- R.2.** $f_{X \Rightarrow Y}(x) = f_Y \circ Apl(x) \circ g_X$, for $x \in O$ and
 $g_{X \Rightarrow Y}(y) = La1(\text{type}(X), g_Y \circ y \circ f_X)$, for $y \in X \Rightarrow Y$.
- R.3.** If $K : \text{Types} \rightarrow \text{Types}$:
 $f_{Q(K)}(x) = \lambda X. f_{K(X)}(Ap2(x, \text{type}(X)))$, for $x \in O$.
 $g_{Q(K)}(y) := La2(r \in O \mapsto g_{K([r])}(y)([r]))$, for $y \in Q(K)$.

6 The Completeness proof.

6.1 Defining logical relations.

In this section we define “candidates” for logical relations on compound types, and then we use them to prove the existence of the 1-1 correspondence S_σ described in proposition 26.

At this stage ρ still denotes any semantic *environment on types* and its extension to all types. We will also need “relational” environments which, essentially, interpret type variables by (f_X, g_X) -relations. To give the precise definition of a relational environment we first define our candidates for compound types.

Definition 31 A candidate is a pair (X, T) such that X is a semantic type, $\text{type}(X) \in \check{O}_{tp}$, and T is an (f_X, g_X) -relation between: the codes of terms with type $\text{type}(X)$, that is $\check{O}_{\#^{-1}(\text{type}(X))}$, and X .

Definition 32 A relational environment Υ on types is a map from type variables to candidates. Let us call pre-relational environment on types a map \mathcal{R} from type variables to O -relations. Then a relational environment can be defined as a pair (ρ, \mathcal{R}) , where ρ is an environment on types, and \mathcal{R} a pre-relational environment such that, for all i , $\mathcal{R}(\alpha_i)$ is an $(f_{\rho(\alpha_i)}, g_{\rho(\alpha_i)})$ -relation on $\check{O}_{\pi(\alpha_i)} \times \rho(\alpha_i)$, where π is the unique environment on types compatible with ρ .

Note : unicity of π is here only up to α -equivalence, as in Definition 19.

Definition 33 The relational environment Υ is compatible with ρ and π , if ρ and π are compatible and ρ (hence π) are the semantic and syntactical environments induced by Υ using the formulas above.

We use the word “compatible” instead of “induced by Υ ” because we will gain some freedom later on, at the level of terms, since Υ will not fix the values of ρ, π on terms

It is useful to note that :

Remark 34 if Υ, ρ, π are compatible, then for any candidate (X, T) , we have that $\Upsilon[\alpha : T]$ is a relational environment and that $\Upsilon[\alpha : (X, T)]$, $\rho[\alpha : X]$ and $\pi[\alpha : \#^{-1}(\text{type}(X))]$ are also compatible.

Example 35 Let Υ_C be the relational environment defined by $\Upsilon(\alpha_i) := (C_i, R_i)$, where C_i is given by Cond 1, R_i is the minimum (f_i, g_i) -relation between \check{O}_{α_i} and C_i , and (f_i, g_i) are also given by Cond 1.

$((f_i, g_i) = (f_{C_i}, g_{C_i}),$ because of the recursive definition of the family (f_X, g_X)).
It should be clear that Υ_C, ρ_C and id are compatible at the level of types.

We now present the formulas which allow to extend uniformly all relational environments to all types. We already know how to extend the ρ -components to all types (or see the Appendix, Section 7).

Definition 36 We define by induction on τ , and at each step for all environment Υ , a binary relation $\mathcal{R}_\Upsilon^\tau$, which only depends on the value of Υ on the free variables of τ . Each Υ will then be extended to all types via: $\Upsilon(\tau) := (\rho(\tau), \mathcal{R}_\Upsilon^\tau)$.

- R1. $\mathcal{R}_\Upsilon^{\alpha_i}$ is the second component of $\Upsilon(\alpha_i)$.
- R2. $\mathcal{R}_\Upsilon^{\sigma \rightarrow \tau}(x, y)$ iff $y \in \rho(\sigma \rightarrow \tau)$ and $\forall v, w (\mathcal{R}_\Upsilon^\sigma(v, w) \Rightarrow \mathcal{R}_\Upsilon^\tau(\text{Ap1}(x, v), y(w)))$.
- R3. $\mathcal{R}_\Upsilon^{\forall \alpha \tau}(x, y)$ iff $y \in \rho(\forall \alpha \tau)$ and for all candidates (X, T) , we have :
 $\mathcal{R}_\Upsilon^\tau[\alpha : (x, T)](\text{Ap2}(x, \text{type}(X)), y(X))$.

Remark that Υ depends only on the applicative behavior of an object. Thus, Υ will not be able to distinguish between η -convertible terms. Apparently, exactly for such a reason, our proof cannot be adapted to a proof of β -completeness.

Lemma 37 For all τ , and for all Υ, ρ, π which are compatible at the level of types, $\mathcal{R}_\Upsilon^\tau$ is an $(f_{\rho(\tau)}, g_{\rho(\tau)})$ -relation between $\check{O}_{\pi(\tau)}$ and $\rho(\tau)$, hence $(\rho(\tau), \mathcal{R}_\Upsilon^\tau)$ is a candidate.

Proof. The proof goes by induction on τ and shows at each step (as usual) that $\mathcal{R}_\Upsilon^\tau$ only depends of the value of Υ (and ρ, π) on the free variables of τ and has domain $\check{O}_{\pi(\sigma)}$. The variable case follows immediately from the compatibility of Υ, π, ρ . Also, the third condition of (f, g) -relations is obviously preserved at each step of induction, so we only have to prove the first two ones.

- Suppose $\tau := \sigma \rightarrow \theta$.

1. Suppose $x \in \check{O}_{\pi(\sigma \rightarrow \theta)}$; we want to show $R_{\Upsilon}^{\sigma \rightarrow \theta}(x, f_{\rho(\sigma \rightarrow \theta)}(x))$.

Suppose we have $\mathcal{R}_\Upsilon^\sigma(v, w)$ for some (v, w) . Then, by induction hypothesis, $v \in \check{O}_{\pi(\sigma)}$ and $v \sim_{tr} v' := g_{\rho(\sigma)}(w)$. Also by induction hypothesis do we have $\mathcal{R}_\Upsilon^\theta(\text{Ap1}(x, v'), f_{\rho(\theta)}(\text{Ap1}(x, v'))$. Since Ap1 commutes with \sim_{tr} and because the third condition is true for $\mathcal{R}_\Upsilon^\theta$, still by induction hypothesis, we have $\mathcal{R}_\Upsilon^\theta(\text{Ap1}(x, v), f_{\rho(\theta)}(\text{Ap1}(x, v'))$. Because of the definition of v' this can be rewritten as :

$\mathcal{R}_\Upsilon^\theta(\text{Ap1}(x, v), [f_{\rho(\theta)} \circ \text{Ap1}(x) \circ g_{\rho(\sigma)}](w))$, hence as :

$\mathcal{R}_\Upsilon^\theta(\text{Ap1}(x, v), f_{\rho(\theta) \Rightarrow \rho(\sigma)}(x)(w))$. The conclusion now follows from $\rho(\sigma \rightarrow \tau) = \rho(\sigma) \Rightarrow \rho(\theta)$.

2. Suppose $\mathcal{R}_\Upsilon^{\sigma \rightarrow \theta}(x, y)$; we want to prove :

$x \in \check{O}_{\pi(\sigma \rightarrow \theta)}$ and $x \sim_{tr} g_{\rho(\sigma \rightarrow \theta)}(y)$.

For all $v \in \check{O}_{\pi(\sigma)}$ we have :

$\mathcal{R}_\Upsilon^\sigma(v, f_{\rho(\sigma)}(v))$ (by induction hypothesis on σ), hence :

$\mathcal{R}_\Upsilon^\theta(\text{Ap1}(x, v), y(f_{\rho(\sigma)}(v)))$ (by definition of $\mathcal{R}_\Upsilon^{\sigma \rightarrow \theta}$).

This forces $\text{Ap1}(x, v) \in \check{O}_{\pi(\theta)}$. A first consequence is that $x \in \check{O}_{\pi(\sigma \rightarrow \theta)}$: just choose $v := \#x^{\pi(\sigma)}$, then v codes a well-formed term of type $\pi(\sigma)$; since $\text{Ap1}(x, v)$ codes a well-typed term of type $\pi(\theta)$ necessarily x codes a well-typed term of type $\pi(\sigma \rightarrow \theta) = \pi(\sigma) \rightarrow \pi(\theta)$.

The second consequence is that

$g_{\rho(\theta)}(y(f_{\rho(\sigma)}(v))) \sim_{tr} \text{Ap1}(x, v)$ (by induction hypothesis on θ).

Since this happens for all v , it follows :

$\text{La1}(\#(\pi(\theta)), g_{\rho(\theta)} \circ y \circ f_{\rho(\sigma)}) \sim_{tr} \text{La1}(\#(\pi(\theta)), \text{Ap1}(x))$ (Remark 14).

Now,

$g_{\rho(\sigma \rightarrow \theta)}(y) = g_{\rho(\sigma) \Rightarrow \rho(\theta)}(y) := \text{La1}(\text{type}(\rho(\theta)), g_{\rho(\theta)} \circ y \circ f_{\rho(\sigma)}) \sim_{tr} \text{La1}(\#(\pi(\theta)), g_{\rho(\theta)} \circ y \circ f_{\rho(\sigma)})$

(compatibility of ρ, π and commutativity of La1 with \sim_{tr}).

On the other hand :

$\text{La1}(\#(\pi(\theta)), \text{Ap1}(x)) \sim_{tr} x$ (Remark 15). Thus

$g_{\rho(\sigma \rightarrow \theta)}(y) \sim_{tr} x$; Q.E.D.

- Suppose $\tau := \forall\alpha.\sigma$.

1. Let $x \in \check{O}_{\pi(\forall\alpha.\sigma)}$. We want to show $\mathcal{R}_{\Upsilon}^{\tau}(x, f_{\rho(\forall\alpha.\sigma)}(x))$,
i.e. $\mathcal{R}_{\Upsilon}^{\tau}(x, f_{\forall Z.\rho[\alpha:Z](\sigma)}(x))$.
So, we have to show $\mathcal{R}_{\Upsilon[\alpha:(X,T)]}^{\sigma}(Ap2(x, type(X)), f_{\forall Z.\rho[\alpha:Z](\sigma)}(x)(X))$
for all candidate (X, T) . By definition of $f_{\forall Z.\dots}$ this amounts to show
that $\mathcal{R}_{\Upsilon[\alpha:(X,T)]}^{\sigma}(Ap2(x, type(X)), f_{\rho[\alpha:X](\sigma)}(Ap2(x, type(X))))$,
which follows from the induction hypothesis applied to σ , and to
 $\Upsilon[\alpha : (X, T)], \rho[\alpha : X], \pi[\alpha : \#^{-1}(type(X))]$
2. Suppose $\mathcal{R}_{\Upsilon}^{\forall\alpha.\sigma}(x, y)$.

We want to show that $x \in \check{O}_{\pi(\forall\alpha.\sigma)}$ and $x \sim_{tr} g_{\rho(\forall\alpha.\sigma)}(y)$, starting from :

$\mathcal{R}_{\Upsilon[\alpha:T]}^{\sigma}(Ap2(x, type(X)), y(X))$ for all candidate (X, T) . (*)

To prove the two parts of this assertion we will use two different instances of (*) which both change a tiny part of Υ using Υ_C .

For the first part we set $(X, T) := \Upsilon_C(\alpha)$, hence $X := \rho_C(\alpha)$. Then $type(X) \sim_{tp} \#\alpha$ and, since $Ap2(x, \#\alpha)$ has to be the code of a well-typed term of type $\pi[\alpha : \alpha](\sigma)$ (by induction hypothesis and Remark 6.3), x can only be the code of a well-typed term of type $\forall\alpha.\sigma'$, with $\sigma' = \pi[\alpha : \alpha](\sigma)$ (up to α -equivalence); now, $\forall\alpha.\sigma' = \forall\alpha.\pi[\alpha : \alpha](\sigma) = \pi(\sigma)$, hence x is the code of a well-typed term of type $\pi(\sigma)$, i.e. $x \in \check{O}_{\pi(\sigma)}$, as required.

We turn now to the second half of the assertion.

Now $g_{\rho(\forall\alpha.\sigma)}(y) := g_{\forall Z.\rho[\alpha:Z](\tau)}(y) := La2(h)$ where

$h := \lambda r : O \ g_{\rho[\alpha:[r]](\sigma)}(y([r]))$.

We have $h(\#\alpha_i) = g_{\rho[\alpha:C_i](\sigma)}(y(C_i))$.

Applying (*) to $\Upsilon[\alpha : \Upsilon_C(\alpha_i)]$, hence to $\rho[\alpha : \rho_C(\alpha_i)]$ and $\pi[\alpha : \alpha_i]$, and applying the induction hypothesis on σ , we get $h(\#\alpha_i) \sim_{tr} Ap2(x, \#\alpha_i)$. Thus $La2(h) \sim_{tr} La2(Ap2(x)) \sim_{tr} x$ (by Remark 15).
Q.E.D.

We end with a substitution lemma, which is needed at the very end.

Lemma 38 *For all strongly compatible Υ, ρ, π , for all α, σ, τ we have :*

$$\mathcal{R}_{\Upsilon}^{\tau[\alpha:\sigma]} = \mathcal{R}_{\Upsilon[\alpha:(X,T)]}^{\tau} \text{ with } (X, T) := \Upsilon(\sigma).$$

Proof. Straightforward induction on τ .

6.2 Relating the various interpretations.

This last subsection is devoted to the statement and proof of the open version of the “top level” lemma that we gave at the beginning.

From now on we deal with three kinds of interpretations, respectively denoted by ρ, π and Υ , but now ρ and π also deal with terms.

ρ denotes any interpretation of $F \perp$ terms and types in the model, it also denotes the environment from which it is issued, and it also denotes its restriction to the world of types. It should be clear from the context what we are really using. By the definition of such interpretations we necessarily have $\rho(t) \in \rho(\sigma)$ if $t : \sigma$.

π denotes a syntactical interpretation of F -terms (and types), and also denotes its restriction to types (cf. Appendix). We necessarily have $\pi(t) : \pi(\sigma)$ if $t : \sigma$.

Υ is the relational interpretation on types defined in Section 6.1.

Definition 39 *We will say that Υ links π to ρ , or that π, ρ and Υ are strongly compatible, if :*

1. Υ, π and ρ are compatible at the level of types, and
2. $\mathcal{R}_\Upsilon^\sigma(\#\pi(x_i^\sigma), \rho(x_i^\sigma))$ for all i, σ .

We have already noticed that the first hypothesis determines the values of ρ and π on types, from that of Υ . If moreover Υ links π to ρ then $\pi(x_i^\sigma)$ is determined, up to $\alpha\beta\eta$ -equivalence, by $\rho(x_i^\sigma)$ and the restrictions of Υ to the type variables which are free in σ (since $\mathcal{R}_\Upsilon^\sigma$ is an O -relation).

The following result is a key one.

Lemma 40 *There exist Υ, ρ such that Υ links id (the identity on types and terms) to ρ .*

Proof. Let $\Upsilon := \Upsilon_C$ and let ρ be the extension of the type environment ρ_C (which was defined via $\rho_C(\alpha_i) := C_i$) by $\rho(x_i^\sigma) = f_{\rho(\sigma)}(\#x_i^\sigma)$. It is immediate to check that Υ links id and ρ (we already know that Υ is compatible with ρ and id at the level of types (Example 35)).

We have already seen that the compatibility conditions on π and Υ w.r.t. ρ propagate to all types (Lemma 22 and 37). The fact that the condition in Definition 39 above propagates to all terms is the desired open version of Proposition 26.

Proposition 41 *If Υ links π to ρ , then for all τ , we have :*

$\mathcal{R}_\Upsilon^\tau(\#\pi(t), \rho(t))$ for all $t : \tau$.

Proof. By induction on the length of t , and at each step for all strongly compatible Υ, ρ, π . Note that, because of the third clause in the definition of (f, g) -relations, and since this does not change the length of t , we may as well replace t by any α -equivalent term in the course of the proof. Note also that we will use this third clause freely in the following proofs.

- The cases where t is a term variable or is of the form uv are immediate, so we deal only with the other 3 cases.

- *term-abstraction.* Suppose $t := \lambda x_i^\sigma . u : \sigma \rightarrow \tau$, with $u : \tau$; by possibly renaming x_i^σ we may also suppose that x_i^σ not free in $\pi(t)$.

The aim is to prove : $\mathcal{R}_{\Upsilon}^{\sigma \rightarrow \tau}(\#(\pi(\lambda x_i^\sigma . u)), \rho(\lambda x_i^\sigma . u))$, and we know that $\mathcal{R}_{\Upsilon'}^{\tau}(\#(\pi'(u)), \rho'(u))$, for all strongly compatible Υ', π', ρ' by induction hypothesis.

Suppose $\mathcal{R}_{\Upsilon}^{\sigma}(v, w)$. Since $v \in \check{O}_{\pi(\sigma)}$, we have $v = \#t'$ for some $t' : \pi(\sigma)$ and our secondary aim is to prove that :

$\mathcal{R}_{\Upsilon}^{\tau}(Ap1(\#(\pi(\lambda x_i^\sigma . u)), \#t'), (\rho(\lambda x_i^\sigma . u))(w))$. Now,

$Ap1(\#(\pi(\lambda x_i^\sigma . u)), \#t') = \#((\pi(\lambda x_i^\sigma . u))t')$ (definition of *Ap1*)

$\sim_{tp} \#(\pi(u[x_i^\sigma : t'])) = \#(\pi'(u))$, where $\pi' := \pi[x_i^\sigma : t']$ and π acts like σ on types. Similarly :

$(\rho(\lambda x_i^\sigma . u))(w) = \rho(u[x_i^\sigma : w]) = \rho'(u)$, where $\rho' := \rho[x_i^\sigma : w]$ and ρ' acts like ρ on types.

Since π, π' and ρ, ρ' coincide on types the compatibility conditions are preserved and the secondary aim follows from the induction hypothesis.

- *type-abstraction.* Suppose $t := \lambda \alpha . u : \forall \alpha . \tau$, with $u : \tau$ and α not free in the type of a term-variable of τ . Since we work up to α -equivalence we may furthermore suppose that α is not free in $\pi(t)$.

The aim is to prove $\mathcal{R}_{\Upsilon}^{\forall \alpha . \tau}(\#(\pi(\lambda \alpha . u)), \rho(\lambda \alpha . u))$, that is to say, to prove $\mathcal{R}_{\Upsilon[\alpha : (X, T)]}^{\tau}(Ap2(\#(\pi(\lambda \alpha . u)), type(X)), (\rho(\lambda \alpha . u))(X))$ for all (X, T) such that $type(X) \in \check{O}_{tp}$ and T is an (f_X, g_X) -relation.

Let σ be such that $type(X) = \#\sigma$.

Set $\Upsilon' := \Upsilon[\alpha : (X, T)]$, $\rho' := \rho[\alpha : X]$, $\pi' := \pi[\alpha : \sigma]$ at the level of types ; these are compatible (at the level of types) since $type(\rho'(\alpha)) \sim_{tp} \#(\pi'(\alpha))$ and nothing is changed for the other type variables. At the level of terms term variables we just have to change π and ρ on those x^θ such that α is free in θ . We set $\pi'(x^\theta) = x_i^{\pi'(\theta)}$ and $\rho'(x^\theta) = f_{\rho'(\theta)}(x_i^{\pi'(\theta)})$, which obviously preserve the strong compatibility condition. Now,

$Ap2(\#(\pi(\lambda \alpha . u)), type(X)) = \#((\pi(\lambda \alpha . u))\sigma)$ (by definition of *Ap2*)

$= \#(\pi[\alpha : \sigma]((\lambda \alpha . u)\alpha)) = \#(\pi'((\lambda \alpha . u)\alpha)) \sim_{tr} \#\pi'(u)$. Similarly :

$(\rho(\lambda \alpha . u))(X) = \rho[\alpha : X]((\lambda \alpha . u)\alpha) = \rho'((\lambda \alpha . u)\alpha) = \rho'(u)$.

Thus, our secondary aim amounts to showing that $\mathcal{R}_{\Upsilon'}^{\tau}(\#(\pi'(u)), \rho'(u))$, which follows by induction hypothesis.

- *application of terms to types.* Suppose $t := u\sigma : \tau[\alpha := \sigma]$, with $u : \forall \alpha . \tau$, and α not free in t and $\pi(t)$. We have to prove :

$\mathcal{R}_{\Upsilon}^{\tau[\alpha := \sigma]}(\#(\pi(u\sigma)), \rho(u\sigma))$. Now,

$\#(\pi(u\sigma)) = \#((\pi(u)(\pi(\sigma))) = Ap2(\#(\pi(u)), \#(\pi(\sigma)))$ (definition of *Ap2*), and $\rho(u\sigma) = \rho(u)(\rho(\sigma))$.

Let us take $X = \rho(\sigma)$. Then $\text{type}(X) \sim_{tp} \#(\pi(\sigma))$ by compatibility of π, ρ . Furthermore $\mathcal{R}_{\Upsilon}^{\tau[\alpha:\sigma]} = \mathcal{R}_{\Upsilon[\alpha:T]}^{\tau}$ where $T = \mathcal{R}_{\Upsilon}^{\sigma}$ (Lemme 38).

The aim follows from the induction hypothesis on u instantiated to $\Upsilon[\alpha : (\rho(\sigma), T)]$, $\rho[\alpha : \rho(\sigma)]$, and $\pi[\alpha : \pi(\sigma)]$.

Proposition 26 now follows easily. Indeed, let t, u be any closed terms of type σ . Then by Lemma 40 and Proposition 41, if $\rho(t) = \rho(u)$ for all ρ , then, for some Υ, ρ we have $\mathcal{R}_{\Upsilon}^{\sigma}(\#id(t), \rho(t))$, $\mathcal{R}_{\Upsilon}^{\sigma}(\#id(u), \rho(u))$. By the properties of (f, g) -relations and $\rho(t) = \rho(u)$ we conclude $\#t = \#id(t) \sim_{tr} \#id(u) = \#u$. We conclude $t =_{\beta\eta} u$, our thesis.

6.3 Appendix on syntactic substitutions.

Such substitutions associates types to types and F -terms (up to α -equivalence) to F -terms, in such a way that $t : \sigma$ implies $\pi(t) : \pi(\sigma)$. They are defined from their restrictions to type and terms variables (or environments), by the inductive rules given below. We denote by π the global substitution, its restriction to types, as well as the environments from which both are issued. As usual $\pi[\alpha : \sigma]$ denotes the substitution on types which behaves like π on all variables different from α , and gives value σ to α . Similarly $\pi[x_i^{\sigma} : t]$ denotes the substitution which behave like π on types variables and on all term variables different from x_i^{σ} , and gives value $t : \pi(\sigma)$ to x_i^{σ} .

The induction rules work on the length of types or terms and define $\pi(\sigma)$ or $\pi(t)$ for all π at each step :

$\pi(\alpha_i)$ is any fixed type, for all $i \in N$,

$\pi(\sigma \rightarrow \tau) := \pi(\sigma) \rightarrow \pi(\tau)$

$\pi(\forall\alpha.\tau) := \forall\alpha.\pi[\alpha : \alpha](\tau)$, if α does not occur free in $\pi(\beta)$ for all $\beta \neq \alpha$ which is free in τ .

$\pi(x_i^{\sigma})$ is any term of type $\pi(\sigma)$

$\pi(tu) := \pi(t)\pi(u)$ and $\pi(u\sigma) := \pi(u)\pi(\sigma)$

$\pi(\lambda x_i^{\sigma}.t) := \lambda x_i^{\pi(\sigma)}.\pi[x_i^{\sigma} : x_i^{\pi(\sigma)}](t)$, if $x_i^{\pi(\sigma)}$ does not occur free in $\pi(x_j^{\tau})$ for all $x_j^{\tau} \neq x_i^{\sigma}$ which is free in t .

$\pi(\lambda\alpha.u) := \lambda\alpha.\pi[\alpha : \alpha](u)$ (recall that, by the formation rules, α cannot be free in the type of a term variable of u).

7 Appendix : interpretation in a model.

Once models are presented as we did, the interpretations of types and terms below and the proof of the related lemmas is routine (a lengthy routine, which has to be lead carefully). One should note however that the relative easiness of these lemmas relies on the fact that we took as much polymorphic functions as possible ; otherwise Lemma 45 below, which asserts the correctness of the interpretations of terms would be more difficult to state and to prove.

In the following *Vartypes* and *Varterms* denote respectively the set of type and term variables, and M is a model in the sense of Section 3.2.

Interpretation of types.

Definition 42 A type-environment is a function ρ from *Vartypes* to *Types*. such that : $\rho(\alpha) \in \text{Types}$ for all type variable $\alpha \in \text{dom}(\rho)$.

Given a type environment ρ and $X \in \text{Types}$, we denote by $\rho[\alpha : X]$ the environment ρ' defined by : $\rho'(\alpha) := X$, and ρ' coincides with ρ everywhere else. We extend the notation to $\rho[\bar{\alpha} : \bar{X}]$, where it is understood that $\bar{\alpha}$ and \bar{X} have the same length (≥ 0), and $\bar{\alpha}$ consists of distinct type variables.

A type-environment can be extended to an interpretation of all F -types by elements of *Types* by induction on the complexity of types, and defining at each step σ , all semantic types $|\sigma|_\rho$ for all type environments ρ .

- $|\alpha|_\rho := \rho(\alpha)$
- $|\sigma \rightarrow \tau|_\rho := |\sigma|_\rho \Rightarrow |\tau|_\rho$
- $|\forall\alpha.\sigma|_\rho := Q(X \mapsto |\sigma|_{\rho[\alpha:X]})$.

To check that $|\cdot|_\rho$ is well-defined we have, as usual, to prove simultaneously, at each step σ , that :

Lemma 43 For all possible $\rho, \bar{\alpha}$, the function $\bar{X} \mapsto |\sigma|_{\rho[\bar{\alpha}:\bar{X}]}$ belongs to $\text{Hom}(\text{Types}^{l(\bar{\alpha})}, \text{Types})$.

This is routine, using that *Univ* is a *c.c.c.* and that \Rightarrow and Q are morphisms.

It is immediate that only the values of ρ on the free variables of σ are of interest.

Interpretation of terms.

Definition 44 An environment (for types and terms) is a function ρ from *Vartypes* \cup *Varterms* to *Types* \cup *Terms* such that the restriction of ρ for types is a type environment, and, for all variable $x^\sigma \in \text{dom}(\rho)$ we have $\rho(x^\sigma) \in |\sigma|_\rho$ (hence $\rho(\sigma) \in \text{Terms}$).

As noted in remark 5 if we stick to models such that $\emptyset \notin \text{Types}$, then we can define term-environments as total functions.

Given an environment ρ , a term variable x^σ and a semantic term $v \in |\sigma|_\rho$, we denote by $\rho[x^\sigma : v]$ the environment ρ' which coincides with ρ everywhere but in x^σ , where $\rho'(x^\sigma) := v$. We extend the notation to $\rho[\bar{x} : \bar{v}]$, where it is understood that \bar{x} and \bar{v} have the same length (≥ 0), and \bar{x} consists of distinct type variables, and v_i is in the semantic type determined by x_i and ρ .

It is not possible to define an environment $\rho[\alpha : X]$, essentially since there is no canonical choice then for $\rho(x^\sigma)$ if α is free in σ . We will however be able

to give a local meaning to the expression $|t|_{\rho[\alpha:X]}$, provided α is not free in a free term variable of t , and more generally to define $|t|_{\rho[\bar{\alpha}:\bar{X};\bar{x}:\bar{v}]}$ if none of the variables in $\bar{\alpha}$ is free in a term variable of t or in one of the term variables in \bar{x} (plus the obvious compatibility conditions).

The interpretation of F -terms under all possible environments is by induction on the complexity of the term t . At each step $t : \sigma$ one

1. Defines $|t|_{\rho}$ for all ρ 's,
2. Check that $|t|_{\rho}$ only depends on the value of ρ on the type and term variables which occur free in t . In particular, if α is not free in any free term variable of t then we can define $|t|_{\rho[\alpha:X]}$ as $|t|_{\rho'}$, where ρ' is any environment such that $\rho'(\alpha) := X$ and ρ' coincides with ρ on all other type variables and on all term variables x^τ such that α is not free in τ . The meaning of $|t|_{\rho[\bar{\alpha}:\bar{X};\bar{x}:\bar{v}]}$ is then the natural one, subject to the obvious conditions..
3. Prove that $|t^\sigma|_{\rho} \in |\sigma|_{\rho}$.
4. Prove Lemma 45 below, so that the definitions at next steps make sense.

The definition goes as follows :

- $|x^\sigma|_{\rho} := \rho(x^\sigma)$
- $|t^{\sigma \rightarrow \tau} u^\sigma|_{\rho} := \text{apl}_{|\sigma|_{\rho}, |\tau|_{\rho}}(|t^{\sigma \rightarrow \tau}|_{\rho})(|u^\sigma|_{\rho})$
- $|\lambda x^\sigma . t^\tau|_{\rho} := \text{lbd}_{|\sigma|_{\rho}, |\tau|_{\rho}}(v \in |\sigma|_{\rho} \mapsto |t^\tau|_{\rho[x^\sigma:v]})$.
- $|t^{\forall \alpha . \sigma} \tau|_{\rho} := \text{Appl}(|t^{\forall \alpha . \sigma}|_{\rho})(|\tau|_{\rho})$
- $|\lambda \alpha . t^\sigma|_{\rho} := \text{Lambda}(X \in \text{Types} \mapsto |t^\sigma|_{\rho[\alpha:X]})$.

Using the abbreviations introduced at the end of Section 3.2, this can simply be written :

- $|x^\sigma|_{\rho} := \rho(x^\sigma)$
- $|t^{\sigma \rightarrow \tau} u^\sigma|_{\rho} := |t^{\sigma \rightarrow \tau}|_{\rho}(|u^\sigma|_{\rho})$
- $|\lambda x^\sigma . t^\tau|_{\rho} := \lambda v^{|\sigma|_{\rho}} . |t^\tau|_{\rho[x^\sigma:v]}$.
- $|t^{\forall \alpha . \sigma} \tau|_{\rho} := |t^{\forall \alpha . \sigma}|_{\rho}(|\tau|_{\rho})$
- $|\lambda \alpha . t^\sigma|_{\rho} := \lambda X . |t^\sigma|_{\rho[\alpha:X]}$.

The lemma we alluded to is :

Lemma 45 For all ρ and all $\bar{\alpha}, \bar{x} := x_1^{\sigma_1}, \dots, x_n^{\sigma_n}$ such that the following makes sense, $(\bar{X}, \bar{v}) \mapsto |t^\sigma|_{\rho[\bar{\alpha}:\bar{X}; \bar{x}:\bar{v}]}$ belongs to $Hom(Types^{l(\bar{\alpha})} \times |\sigma^1|_{\rho} \times \dots \times |\sigma^n|_{\rho}, Terms)$.

Once more all this is (a careful and lengthy) routine, using that we are within a *c.c.c.* and that we made enough care with our domains and codomains of morphisms.

Then we are able to prove :

Lemma 46 $\sigma =_{\alpha} \tau$ implies $\mathcal{M} \vDash |\sigma|_{\rho} = |\tau|_{\rho}$ for all environment ρ .

$t =_{(\alpha)\beta} u$ implies $\mathcal{M} \vDash |t|_{\rho} = |u|_{\rho}$ for all environment ρ .

If \mathcal{M} is extensional, then the last sentence is also true for $\beta\eta$ -equivalent terms.

The proof is once more straightforward and the case of immediate reduction is done using that all pairs $(lbd_{X,Y}, appl_{X,Y})$, and $(Lambda, Appl)$ are retraction pairs (plus the adequate variation for the extensional case).

References

- [1] M. Abadi, G. D. Plotkin, A logic for parametric polymorphism, in TL-CA'93, SLNCS n°664, p.361-375, Springer 1993.
- [2] R. Amadio, K. Bruce, and G. Longo, The finitary Projection Model for 2nd Order Calculus, IEEE Conference on Logic in Computer Science, Boston, Juin 1986.
- [3] A. Asperti and G. Longo, Categories, Types and Structures : an Introduction to Category Theory for the working Computer Scientist, Cambridge, Mass., MIT Press, 1991.
- [4] H. Barendregt, The λ -calculus, its syntax and semantics, Studies in Logic vol.103, North-Holland, revised edition 1984.
- [5] F. Barbanera, S. Berardi, "A domain of domains model of polymorphism", technical report, University of Turin, 1997.
- [6] S. Berardi, Retractions on dI-Domains as a Model for Type:Type, Information and Computation 94, p.204-231, 1991.
- [7] S. Berardi, Ch. Berline, Building continuous webbed models for System F , preprint, available end of June 98.
- [8] S. Berardi, Ch. Berline, "Notes on models of system F ", Technical Report, Université de Paris VII, en préparation.
- [9] C. Berline, Rétractions et Interpretation Interne du Polymorphisme : le Problème de la Rétraction Universelle, Informatique théorique et Applications/Theoretical Informatics and Applications vol.26, n°1, p.59-91, 1992.

- [10] C. Berline and K. Grue, A κ - denotational semantics for map Theory in ZFC+SI, Theoretical Computer Science 179 (1997) , p.137-202.
- [11] K. Bruce, A.R. Meyer, and J.C. Mitchell, The semantics of 2nd order λ -calculus, Information and Computation 85 (1990), p.76-134.
- [12] T. Coquand, C. Gunter, and G. Winskel. Domain theoretic models of polymorphism, Information and Computation, vol.81, 123-167, 1989.
- [13] P. Di Gianantonio, F. Honsell and G. Plotkin, Uncountable limits and the lambda-calculus, Nordic Journal of Computing 2(1995), 126-145.
- [14] R.C. Flag, K-continuous Lattices and Comprehension Principles, for Frege Structures, Ann. of Pure and Applied Logic 36, p.1-16, 1987.
- [15] H. Friedman, Equality between functionals, In R. Parikh, editor, Logic Colloquium 1972-1973, Lecture notes in Mathematics 453, Springer-Verlag, New-York, 1975.
- [16] J.Y. Girard, Une extension de l'interprétation de Gödel à l'analyse, et son application à l'élimination des coupures et à la théorie des types. Proceedings of the 2nd Scandinavian Logic symposium, J.E. Fenstadt, ed.,p.63-92, North Holland, 1975.
- [17] J.Y. Girard, The system F of variable types, fifteen years later, Theoretical computer Science, vol.45, p.159-192, 1986.
- [18] G. Longo, Parametric and Type-dependent Polymorphism, Fundamenta Informaticae, vol. 22, 1-2, p.69-92, 1995.
- [19] G. Longo, K. Milsted and S. Soloviev, The genericity theorem and the notion of parametricity in the polymorphic λ -calculus, Theoretical Computer Science 121, p.323-349, 1993.
- [20] N. McCracken, A Finitary Retract Model for the Polymorphic λ -calculus, unpublished, 1984.
- [21] P. O'Hearn, Parametric Polymorphism, in: M. P. Fiore, A. Jung, E. Moggi, P. O'Hearn, J. Riecke, G. Rosolini and I. Starked, Domains and Denotational Semantics : History, Accomplishments and Open Problems, Bulletin of EATCS, vol.59, p.227-256, 1996.
- [22] G.D. Plotkin, Lambda-definability and Logical Relations, Technical report SAIRM-4, School of artificial intelligence, University of Edinburgh, 1973.
- [23] G. D. Plotkin, Lambda-definability and the full-type hierarchy, in J.P. Sel-din and J.R. Hindley, eds., To H.B. Curry : Essays on Combinatory Logic, Lambda-calculus and Formalism, Academic Press, New-York, 1980.

- [24] G.D. Plotkin, A power domain for countable non-determinism, in : Lecture Notes in Computer Science vol. 140, ICALP'82, p.418-428, Springer Verlag, Berlin, 1982.
- [25] J.C. Reynolds, Towards a Theory of Type Structure. In Colloque sur la Programmation, LNCS 19, p.408-425, Springer-Verlag, 1974.
- [26] J.C. Reynolds, Polymorphism is not set- theoretic, in : Mathias and Rogers eds., Cambridge Summer Scool in mathematical Logic, Lecture Notes in Mathematics 337, p. 232-252, Springer Verlag 1973.
- [27] A. Simpson, Categorical completeness results for the simply-typed λ -calculus, in : M. Dezani-Ciancaglini and G. Plotkin eds., Lecture Notes in Computer Science vol. 902, TLCA'95, p.414-427, Springer Verlag, Berlin, 1995.
- [28] D. Scott, Data Types as Lattices, S.I.A.M. J. of Computing 5, p. 522-587, 1976.
- [29] D. Scott, A Space of retracts, manuscript.
- [30] R. Staman, Completeness, Invariance and λ -definability, J. of Symb. Logic, vol.47, n°1, 1982.
- [31] A.S. Troelstra, Notes on second order arithmetic, in : Mathias and Rogers, eds., Cambridge Summer Scool in mathematical Logic, Lecture Notes in Mathematics 337, p. 171-205, Springer Verlag 1973.

S. Berardi,

Dipartimento di Informatica,
 Università degli Studi di Torino,
 Corso Svizzera 185,
 10149 Torino, ITALIA.
e-mail: stefano@di.unito.it.

C. Berline,

Equipe de Logique mathématique (case 7012),
 CNRS-Université Paris7,
 2 place Jussieu,
 75251 Paris cedex 05, FRANCE.
e-mail: berline@logique.jussieu.fr