Beta-eta-complete models for System F

Stefano Berardi, Chantal Berline

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

Stefano Berardi & Chantal Berline.

Final version, January 27th, 2002

Abstract

We show that Friedman's proof of the existence of non-trivial $\beta\eta$-complete models of $\lambda_{\rightarrow}$ can be extended to system F. We isolate a set of conditions which are sufficient to ensure $\beta\eta$-completeness for a model of F (and α-completeness at the level of types), and we discuss which class of models we get. In particular, the model introduced in [5], having as polymorphic maps exactly all possible Scott continuous maps, is $\beta\eta$-complete and is hence the first known complete nonsyntactic model of F. In order to have a suitable framework in which to express the conditions and develop the proof, we also introduce the very natural notion of "polymax models" of System F.

1 Introduction.

In this paper we contribute to the study of nontrivial models of the notion of polymorphic maps of System F ([16],[29]). This study was started in Girard [16], Scott [33] and McCracken [22] [23], Reynolds [30], Girard [18], and continued by Amadio-Bruce-Longo [2], Coquand-Gunter-Winskel [12] and Berardi [6].

Our main concern is to prove that there exists a large class of nontrivial $\beta\eta$-complete models of F. Essentially, this class consists of models having (implicit) primitive types for basic data (integers, etc.), and "overloaded" functions, that is, functions whose actual definition depends on the type of the argument, or functions having a case definition over types. In particular, we will prove that the model built in [5] is $\beta\eta$-complete. This model, called here the BB-model for short, belongs to Scott's semantics, and the polymorphic maps in it are exactly the Scott continuous ones; furthermore, its definition is very concrete and its construction makes no reference either to the syntax of F or to the primitive types and overloaded functions mentioned above.

We will isolate a set of conditions about models of F, formally translating the requirement: "there exist primitive data types plus a case map on types", 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. We will notice that the conditions also force that one is dealing with (implicit) models of $F_{\omega}Y$ (Girard's

F_ω plus recursion on types and terms) but that they do not imply completeness for this wider setting.

Independently of the mathematical interest of the existence of a nonsyntactical $\beta\eta$-complete model of F, we think that there is an interest for theoretical computer scientists to consider models which, like the BB-model, satisfy the above requirement. This is developed in [5]; to summarize: it enables us to model computations really depending on the "type tag" of their inputs, a situation which often arises when one mixes functional and imperative features (think of polymorphic printing, order tests, sorting maps, and so on) or mixes classes from object-oriented languages with second-order calculus. Incidentally, note that it is then an immediate consequence of the completeness of the BB-model that all the extensions mentioned above are "conservative over F", in the sense that they do not force new equations between F-terms.

The BB-model is generalized in [7] to a whole class of feasible models of F which we call "webbed models of System F". Some of our formal conditions are satisfied in all of these models, and it is very easy to check for most of the other ones whether they are satisfied or not in a specific webbed model. This class belongs to the continuous semantics (and, in contrast to previous modelizations of F, it admits no analogue in the stable semantics). The class contains infinitely many other $\beta\eta$-complete models, which are powerful enough to model various extensions of F, in the sense above.

In contrast, none of the previously known extensional models of System F satisfies all our conditions for completeness, and most are known to be incomplete: the only exception is that of universal retraction models for which the question is left open; this is discussed in the next section and in section 5.

In order to have a reasonably general framework to express our conditions, we introduce in this paper a very natural notion of models of System F, which we call "polymax models of F". The class of polymax models includes in particular all the webbed models of [7] and the universal retraction models and it allows for a very transparent interpretation of F-terms. We think that it should also be of interest for other works on polymorphism since it covers a great variety of concrete models which can easily be forced to satisfy domain equations and inequations, as already shown by some examples in [7]. However, as suggested by its name, this abstract class covers only models with "the maximum number of polymorphic maps" and hence rules out all the models of F which have constrained sets of polymorphic maps, and in particular parametric models. This does not matter here since all such known (nonsyntactic) models are indeed incomplete.

The construction of the BB-model and the nature of webbed models is of no use for the completeness result and its proof. However, some familiarity with them will help the reader to understand better the examples which give

substance to the notion of polymax model as well as our discussion about the possible satisfaction of the completeness conditions. So we give a brief introduction to these models in Appendix C.

We now describe the plan of the paper.

In Section 2 we sketch the problematics about polymorphic maps, about their models, and then the problem of $\beta\eta$-completeness. Finally we list the questions we leave open.

In Section 3 we recall the syntax of system F.

In Section 4 we define the notion of polymax models of System F and we discuss it briefly with respect to a few more established definitions or notions of models of system F. The interpretation of System F in polymax models is presented explicitly in Appendix A [1]. Appendix B makes explicit the fact that polymax models fit the Bruce-Meyer-Mitchell description [11] of second-order models and that the PER-models are not polymax, and finally compares the notion of polymax model with the two categorical notions of models for system F proposed in [3] and in [25]. Appendix C gives a flavor of the webbed models and of the BB-model.

In Section 5 we present and discuss our set of conditions, divided into two groups, and their possible satisfaction in concrete models.

In Section 6 we prove a general recursion lemma which is our basic tool for constructing polymorphic maps, and which encapsulates nearly all the conditions of the second group.

In Section 7 we give an overview of the completeness proof.

Sections 8 to 11 present the four major steps in the development of this proof. Most of the local proofs are postponed to Appendix D and F. Appendix E deals with the precise definition of syntactic interpretations.

Acknowledgments. We are grateful to the referees and to the editor for their constructive comments, which in particular resulted in the addition of the overview Section 7, of several appendices, and of many remarks. Thanks also to G. Rosolini for fruitful discussions, and to Russ Harmer who kindly polished our English in the first version.

2 Problematics about polymorphic maps and $\beta\eta$-completeness.

In this section we suppose the reader already to be 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 problematics about them. Call "type constructor" any function $\Phi : Types \to Types$, where $Types$ is the class of types, and types

[1] The first version of this paper contained also the proof that the intended interpretation in polymax models was correct. This proof has now been moved to [7].

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 $Id(\alpha) = (\alpha \to \alpha)$. The polymorphic map *id* takes as input α, and returns the identity $id(\alpha)$ of type $(\alpha \to \alpha)$.

The type of the polymorphic maps associated to Φ is denoted $\forall\alpha.\Phi(\alpha)$. System F 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 of the set-theoretical cartesian product $\Pi\alpha \in Types.\Phi(\alpha)$, 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 nonempty). However the most natural attempt fails : it is indeed impossible to expand the standard model of simply typed λ-calculus (which interprets $\alpha \to \beta$ as the full set of functions from α to β) into a model of F, as Reynolds [30] shows [2]. Therefore, for interpreting F (via functional models), one necessarily has 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 then explain how the BB-model takes care of polymorphism.

Models of polymorphic maps. In the simplest realizability or PER models of F, which originate in Girard and Troelstra [16],[35], polymorphic types are empty or realized by (essentially) constant maps. Thus, a PER model, while it is a useful tool for studying the properties of F as a programming language, is too restrictive for studying polymorphism in full generality. This last sentence also applies to "parametric" models, which require that polymorphic maps behave as uniformly as possible (see [24] for a survey). Hence it should be clear that we will not be concerned here with these kinds of models.

The first models with nonconstant polymorphic maps, which we will call here *universal retraction models* or *u.r.* models, were introduced, for the continuous semantics, by D. Scott and McCracken in [32][3], [33], [23] and further studied 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 (suitable) retractions of the model, and are 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 the continuous maps f which send any $\alpha \in Types$ to some $f(\alpha) \in \Phi(\alpha)$. 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 limitation of these constructions was the complexity in the definition of the set $Types$. This complexity has forbidden, until now, any extensive study of their properties.

Then a categorical model was introduced by Girard in [18]. Girard interprets $Types$ as the category of coherent domains and embeddings, and type construc-

[2] Reynolds's theorem is however no longer true in intuitionistic set-theory [25].

[3] This paper does not deal with models of System F but it is the very paper which introduces retractions in connection with types.

tors as "continuous" and "stable" functors over $Types$. Here "continuous" and "stable" are categorical notions, analogous to the classical ones. The model is universal in the sense that second order quantification was over all coherent domains, and not only over subdomains of a given domain. Moreover the model is economical in some sense[4], and it interprets polymorphic maps f associated to a type constructor Φ as all the families of elements $f(X) \in \Phi(X)$, indexed over $X \in Types$, which satisfy the following requirement: for each embedding $j : X \to Y$, $f(X)$ is the "trace of $f(Y)$ over $\Phi(j)$" (in a sense made precise in [18]).

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

Then Coquand-Gunter-Winskel [12] (CGW for short) showed that Girard's construction could be carried out in the category of Scott domains and embeddings, dropping all the "stability" requirements. They interpreted type constructors as "continuous" functors (again, in a categorical sense).

By building on 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), and more generally in all the webbed models, $Types$, 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" [5] $Terms$ 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 $f : Types \to Terms$, such that $f(X) \in \Phi(X)$ for all $X \in Types$. Here no category theory is needed (when dealing with any specific model) and $Types$ is completely understood.

The interpretation of types and terms in webbed models is simpler than in all previous models, except maybe for the simplest realizability models (for a comparison with universal retraction models see [7]).

The BB-model (and all webbed models) might be further generalized taking κ-continuous maps, for κ any regular cardinal [6]. It is not clear how far we might go in adding polymorphic maps to the BB-model.

[4] For example $\forall X.\forall Y.X \to Y$ gets a trivial interpretation (the one element coherent space) and the interpretation of the data type of Booleans $\forall X.(X, X \to X)$ has 4 elements. For more complex data types the situation is less simple.

[5] In the case of the BB-model, or more generally of all the models of [7], (the carrier of) $Terms$ is exactly the union of the (carriers of) the elements of $Types$.

[6] The κ-Scott topology is a weakening of Scott topology (case $\kappa = \omega$), which admits more and more monotone functions as κ increases. The ω_1-case was introduced by Plotkin [28] for modelling parallelism(see [13]) and greater cardinals are used for foundational purposes ([14],[10]).

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

To prove this is our contribution to the topic.

We will now explain what we mean by $\beta\eta$-completeness, why we got interested in finding complete models, and why most known models are incomplete.

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

For *typed λ-calculus*, the problem has been tackled by H. Friedman [15] (and then A. Simpson [31]), who solved the *simply typed* case (see below). Concerning *untyped λ-calculus*, the problem of finding a nonsyntactical complete model has only been solved recently, by Di-Gianantonio, Honsell, Plotkin [13]; several interesting related questions being however left open by their result and proof (see below). For F it was completely open.

A further motivation: Axiom C and Genericity. Besides the natural mathematical and logical motivations above, we also got interested in this question via the problem raised by Longo of finding a complete model for FC, the equational theory studied by Longo, Milsted and Soloviev in [20]. FC is defined as the equational theory of F plus "Axiom C". This axiom, which is proved in [20] to be independent of F and F_η, is indeed a scheme which expresses a very natural, but rather weak, notion of "uniformity of proofs". Axiom C says 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 for all terms t of type $\forall\alpha.\sigma$.

It was natural then to look first at the existence of complete models for F [7].

For the discussion below we note that, since Axiom C is equational and independent of F and $F\eta$, a model of FC cannot be complete for F or $F\eta$.

It is also interesting to note that "parametricity" (for example, Reynolds' "relational parametricity"), which is a strong notion of "uniformity of proofs", implies FC, but that the converse is false (Girard's model below, for example, satisfies FC but is not relationally parametric (see [24])).

Negative examples of completeness. First there are obvious sources of incompleteness linked to the lack of polymorphic maps in a model (this is related to the possible partial parametricity of the model). The simplest of them is the following: suppose that the interpretation of some 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. It is also clear that a model where all the polymorphic maps

[7] We had the intuition that a complete model of FC could be found as a stable analogue of the BB-model (no model of the class developed in [7] for the continuous semantics can satisfy Axiom C). However we know now that there is a conceptual obstacle to the development of a stable analogue of our webbed models (cf.[7]).

associated to a constant constructor Φ are themselves constant, will satisfy axiom C and hence will not be complete for F or F_η; this is the case for the PER models and for Girard's model, for example. Finally we note that Girard's and PER models are not complete for FC (and do not satisfy genericity either); this also follows very easily from the fact that there are closed types which are interpreted by singleton sets in them (see [19]).

We now give another source of incompleteness, which is also linked to a lack of polymorphic maps. Let N be the F-type of integers (as defined in [18]). 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 $P\omega$). Then the model equates two closed F-terms $f, g : N \to N$ if and only if $f(n) =_\beta g(n)$ for all closed $n : N$. Such a model equates for example left and right addition (and similarly with left- and right- *or)*, which are clearly non $\beta\eta$-convertible since they are normal and distinct. These two terms are observationally equivalent [8] but are rather easily seen to be distinguished by all models whose types possess bottom elements (e.g. Girard's model and *u.r.* models).

Positive examples of completeness.

Friedman [15] showed that all full models of simply typed λ-calculus which include integers in the base type are $\beta\eta$-complete (where "full" means that $\sigma \to \tau$ is interpreted as the set of *all* set-theoretic 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 nonconvertible 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 nonconvertible terms by applying them to "very generic" maps, built by the Axiom of Choice (of Set Theory).

Another possibility for having completeness was discovered by Simpson [31], by building on a syntactic result of Statman [34]. Simpson proved that all models of simply typed λ-calculus, which include 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 λ-calculus inside 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 such maps enabling us to encode the term model.

As a matter of fact, when proving that the BB-model was complete we also discovered that it contained some polymorphic maps able to produce such

[8] Since, by a result of Statman (public electronic communication, 1986), two F-terms t and u are observationally equivalent iff $t = u$ is consistent with $\beta\eta$-equivalence.

an encoding (see 7.2, Points 5 and 6). In particular, we use polymorphic case functions testing the shapes of types (Cond.II.4-5); such maps do not exist in, for instance, PER models. In fact, the work done in the present paper proves that the existence of such a (polymorphic) discrimination over types is sufficient to imply the maximal discrimination over terms, provided that $Types$ is rich enough, well structured, and that we have some standard computation tools available.

Remarks and open questions related to the completeness problem.

- We treat $\beta\eta$-completeness here since our proof relies on some kinds of "logical relations", which, as for usual logical relations, do not allow us 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.

- We conjecture that no model of F in the stable semantics can be $\beta\eta$-complete. In any case, a proof of existence of such complete models should be based on different arguments than the ones presented here, for example because of the lack of the polymorphic *case function* in the stable case.

- Simpson [31] proved completeness for all models of simply typed λ-calculus including integers, sum and product, by proving that they are complete for terms of type T, where T is the type of binary trees. Then he used a result of Statman [34] 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 any model complete for terms of type σ is also complete for the terms of all types (but we know that this is false when σ is the polymorphic type of binary trees).

- Also, it is a long-standing 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 know whether the universal retraction models can be complete.

- As mentioned in the introduction, all the webbed models, and in particular the BB-model are in fact (implicit) models of $F_\omega(Y)$, but the completeness proof doesn't extend to F_ω, for a rather strong reason which makes us conjecture that no polymax model can be complete for F_ω. Since most of

the other known nonsyntactical model are not even complete for F, it is quite possible that no such model exists at all.

- We do not pretend of course that the set of conditions for completeness that we give in Section 5 is optimal, and we hope that a more elegant, and maybe more general, set of conditions will be provided by future work.

3 System F.

In this section, we recall the syntax of System F (following [18]).

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, ... are types ($i \in N$).
- $\sigma \to \tau$ is a type if and only if σ and τ are types,
- $\forall\alpha.\sigma$ is a type iff α is a type variable and σ is a type; the variable α is bound in $\forall\alpha.\sigma$.

Except when we encode the syntax, types will be considered only up to α-equivalence (renaming of bound variables) and $\sigma[\alpha : \tau]$ will then denote the type obtained by correct substitution of the type τ for all the 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^σ, y^σ, ..., where σ can be any type, are terms of type σ.
- $\lambda x^\sigma.t$ is a term of type $\sigma \to \tau$ iff t is a term of type τ ; the variable x^σ is bound in $\lambda x^\sigma.t$.
- tu is a term of type τ iff, for some σ, t is a term of type $\sigma \to \tau$ and u is a term of type σ.
- $\lambda\alpha.t$ 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 bound 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 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 for all free occurrences of x^σ within t. We

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

Immediate β-reduction (or reduction of β-redexes) is defined by :
$(\lambda x^\sigma.t)u^\sigma \rightarrow_\beta t[x^\sigma : u^\sigma]$ and
$(\lambda\alpha.t)\sigma \rightarrow_\beta t[\alpha : \sigma]$ (recall that α is not free in a free term variable of t).
Immediate η-reduction (or reduction of η-redexes) is defined by :
$\lambda x^\sigma.tx^\sigma \rightarrow_\eta t$ if $t : \sigma \rightarrow \tau$ for some τ and x^σ is not free in t.
$\lambda\alpha.t\alpha \rightarrow_\eta t$ if $t : \forall\alpha'.\sigma$ for some α', σ and α is not free in t.
Immediate $\beta\eta$-reduction is the union of both and is denoted by $\rightarrow_{\beta\eta}$.

We define *β- (resp. $\beta\eta$-) reduction,* still denoted by $\rightarrow_{\beta\eta}$, as the smallest transitive relation containing the corresponding immediate reduction, and compatible with the formation of terms. Both reductions are Church-Rosser and strongly normalizing [17]. 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 $=_{\beta\eta}$.

Notation 1 *If t is a term then $nf(t)$ will denote its $\beta\eta$-normal form, namely the unique normal term to which t $\beta\eta$-reduces. To be able to uniformly state later a result concerning both terms and types, we set $nf(\sigma) := \sigma$ if σ is a type.*

4 Polymax models of F.

In this section we isolate a restricted but transparent notion of models of F, which we call *polymax models* of F.

This framework is new, as far as we know. It is less general but much easier to grasp and to work with than any other previous one; in particular our definition requires no more category theory than knowing what a cartesian closed category (c.c.c.) is (functors are not used here). Finally it is particularly significant for giving a global description of the kinds of models and properties we are interested in.

We also tried to fix notations which are as meaningful as possible.

The definition of polymax models is abstract enough to fit several different settings (for example universal retraction models [9], the (disjoint) class in [7], and all models of untyped λ-calculus when viewed as trivial models of F), but it does not accommodate the PER models, nor Girard's and CGW's models; as we have already mentioned, this is not a drawback here since these models have been shown to be incomplete. In contrast, our definition covers nonextensional models, even if the bare completeness proof here only works for F_η. One reason is that we hope that the result could be used to prove that some models of [7] have a theory included in F_η, and would allow us to isolate good candidates for being

[9] In fact it only covers the universal retraction models which are based on closures or projections. A slight generalization is needed to include the models based on finitary retractions, which is presented in [7].

complete for F. This choice is not at all costly: one just finds here retraction pairs where one would find pairs of inverse isomorphisms in the extensional case.

Very general definitions/notions of models of F have already been proposed in the literature[10], namely: the definition of Bruce-Meyer-Mitchell [11], and a few categorical definitions (cf. Asperti-Longo [3, Ch11], and in particular p.276 for references to Moggi, Seely and al.). It is also worth mentioning Pitts' categorical definition of topos models of F [25]. Since these definitions are "well-established", it seems we have to be more explicit about the reasons why we do not start from one of them.

The reason why we do not start from a general categorical definition, whether well known or possibly adapted for our paper, is that such definitions are hermetic to the nonspecialist, and that their high level of abstraction is of no use here. The reason why we do not start from the definition of [11] is different. This definition describes in a rather natural way what a second-order structure is and what the interpretations of types and terms in it should be, but then it only defines a model to be a second-order structure such that these interpretations are well-defined, without giving any information on the way one could succeed in doing so. In contrast it is a straightforward theorem that the definition of polymax models *really depicts models* [11] of System F [7], and, then, it is an easy observation that they are also models in the sense of Bruce-Meyer-Mitchell (cf. Appendix B). Due to the intrinsic complexity of the "well-established" categorical definitions of models of F, it is far less evident to check whether polymax models fit such a definition. In Appendix B we compare the definition of polymax models to Moggi's general definition of the "internal models" of F (cf. [3]) and Pitts' definition of topos models. Our conclusion is the following. First, there is no general way to view polymax models as internal models, although internal and polymax models interpret formulas in the same way. Note that this does not contradict the fact that universal retraction models can be described either as internal models or, much more easily, as polymax models [7]. Second, the definition of topos models of F is rather alien to the definition of polymax. Finally none of these categorical definitions is adequate for describing the webbed models of [7], and in particular the BB-model.

Finally, it is plausible that the definition of polymax models can be turned into a categorical one, but this would largely increase its level of complexity (cf. Appendix B) and we claim that it is not worth the effort [12].

[10] Due probably to the difficulty of writing out a significant general definition of models, several famous papers which build models of F (e.g. [18]) do not put these models within a general context.

[11] Is also true of the categorical definitions, of course, that they do produce models, and that they embody the deep structural reasons why this is true.

[12] Furthermore, since our aims are rather practical, the possibility of working with genuine sets, and genuine set inclusions $i_X : X \hookrightarrow Terms$ is a blessing.

4.1 Preliminaries.

Each polymax model will be parameterized by a cartesian closed category with enough points [4, p.108], called $Univ$, and by other parameters which will be objects, morphisms, and families of morphisms of $Univ$. It should be clear that the same category $Univ$ can give rise to infinitely many models of F (see [7]). The reader can think of $Univ$ as being the category of Scott domains and continuous functions (which is indeed a suitable context for the BB-model and will be called here *Scott's c.c.c.* for short), and more generally as being any category whose objects are domains [13] and morphisms are continuous functions (possibly submitted to further conditions).

We emphasize that our intention here is not to give a categorical framework, but rather to give a convenient mathematical framework using the terminology of cartesian closed categories.

In particular, we will always assume that the objects of $Univ$ have a carrier set and that $Hom(A, B)$ is a subset of the set of all functions from the carrier set of A to the carrier set of B [14]. It will be convenient to speak of a *subset* D of an object A or of an *element* x of A. By this we understand the naive interpretation: we forget everything about A except that its carrier is a set. Thus we will use freely "$x \in A$".

The objects of $Univ$ will be denoted A, B, X, Y, Z.

If $Univ$ is some usual category of domains, then each A is (in particular) a partially ordered set with a bottom element, denoted by $\perp_A$.

The crucial semantic notion for modelling β (resp. $\beta\eta$) equivalence, independently of the choice of the formal framework, is that of a retraction pair of morphisms (resp. of inverse isomorphisms). We recall the definition, and fix notations in the meanwhile.

Notation 2 *We write:* $(f, g) : A \rightleftarrows B$ *to mean that* (f, g) *is a pair of morphisms* $f \in Hom(A, B)$ *and* $g \in Hom(B, A)$.

Definition 3 *A* retraction pair *is a pair* $(f, g) : A \rightleftarrows B$ *such that* $g \circ f = id_A$. *We then say that* A *is a* retract *of* B *and that* f *is* left invertible. *If furthermore* $f \circ g = id_B$ *then* f *and* g *are* inverse isomorphisms *between* A *and* B.

The definition of polymax models uses the following definition.

Definition 4 *Let Univ be a category whose objects are sets (with additional structure) and whose morphisms are (specific) functions. We say that the nonempty object* X *is a* substructure *of the object* A *if* X *is a subset of* A *and, for all pairs* (Z, f) *such that* Z *is an object and* f *is an ordinary function from* Z *to* X, *we have that* f *is a morphism iff* $i \circ f$ *is a morphism, where* i *is the inclusion map (this implies trivially that* i *is a (mono-)morphism).*

[13] In the generic sense of: directed complete partial orders (cpo's) and, possibly, additional structure (ex: Scott domains).

[14] A c.c.c. with enough points is a concrete category, so this is not a real restriction.

For example, if $Univ$ is a category of Scott domains and continuous functions, the second condition in Definition 4 expresses the fact that the order of X is the restriction of that of A, and that for all nonempty directed subsets D of X, the sup of D in X remains the sup of D in A. In the stable semantics one should add a similar condition for suitable $inf s$, and so on.

4.2 A transparent way of modelling F.

By a *polymax model of system* F we mean here [15] a tuple

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

such that $Univ$ is a category, $Types$ 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 $Univ$. This tuple is subject to the requirements listed below.

1. We assume that $Univ$ is a cartesian closed category with enough points. We assume furthermore that the objects of $Univ$ have *nonempty* carrier sets, and that $Hom(A, B)$ is a subset of the set of all functions from the carrier set of A to the carrier set of B .

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

   It is clear that all these assumptions are satisfied by Scott's *c.c.c.*

2. $Types$ and $Terms$. The intention is that F-types are interpreted by elements of $Types$ and (explicitly typed) F-terms by elements of $Terms$. By analogy with the syntax, the elements of $Types$ are called *semantic types* and are denoted by X, Y, Z.

   We require that the elements of $Types$ (when $Types$ is viewed as a mere set) are objects of $Univ$, and furthermore are substructures of the object $Terms$ [16].

3. $\Rightarrow \in Hom(Types \times Types, Types)$ is (required to be) a *morphism* and it will be taken as the interpretation of the arrow constructor on syntactic types.

4. lbd and apl are indexed by $Types \times Types$. The family (lbd, apl) will allow us to interpret abstraction over a term variable, and application of a term to a term.

[15] As noticed before, the definition is mildly generalized in [7], for reasons which are not relevant here.

[16] If $Univ$ is Scott's c.c.c., then $Types$ appears as a "domain of domains": it is a domain and the elements of the underlying set are domains. We will give a very concrete example in Example 7 below.

We require that, for all $X, Y \in Types$, the pair $(lbd_{X,Y}, apl_{X,Y})$ is a retraction pair making $X \to Y$ a retract of $X \Rightarrow Y$.

5. $Q \in Hom((Types \to Types), Types)$.

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

6. We require that (*Lambda*,*Appl*) is a retraction pair making $Types \to Terms$ a retract of $Terms$.

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

7. For all $F \in Hom(Types, Types)$ we define $Hom_F(Types, Terms)$ as 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 $Q(F)$ (recall that $Q(F)$ is a substructure of $Terms$). In other words:

   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 $Appl(u)(X) \in F(X)$.

Remark 5 *We emphasize that $X \to Y$, and more generally $A \to B$, is never a semantic type in our setting, while $X \Rightarrow Y$ is.*

Remark 6 *It is clear that Conditions $6 + 7$, which express in particular that the set $Hom_F(Types, Terms)$ is "contained in" $Q(F)$, are the very conditions that force all "possible" polymorphic functions to be in $\forall X.F(X)$ and which rule out all the models with constrained sets of polymorphic maps. Moreover Conditions $6+7$ express that the previous embedding is "uniform in F". Finally they imply that $Hom_F(Types, Terms) = Hom_{F'}(Types, Terms)$ if $Q(F) = Q(F')$. However they do not seem to imply that Q is injective, although this is true in all the known polymax models.*

Out of any polymax model $\mathcal{M}$, 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 compatible way, we may define an interpretation $[.]_\rho$ of F-types and F-terms (see Appendix A). The fact that this interpretation is well defined for all polymax models follows from a lemma whose statement is quite simple and whose proof is straightforward but uses crucially the fact that we do not constrain polymorphic maps (see [7]).

All polymax models of F identify $(\alpha)\beta$- convertible terms and α-convertible types. A polymax model identifies η-convertible terms, and is then labelled *extensional*, if all pairs $(lbd_{X,Y}, apl_{X,Y})$ are pairs of inverse isomorphisms, and, for all $F \in Types \to Types$, the restriction of $Lambda \circ Appl$ to $Q(F)$ is the identity.

Example 7 .

1. *"Trivial models". Let (M, lbd, apl) be a model of untyped λ-calculus in a c.c.c. $Univ$ (which means that (lbd, apl) is a retraction pair making $M \to M$ a retract of M). We let $Types := \{M\}$, $Terms := M$. Thus $(M \Rightarrow M) = M$ and $Q(F) = M$ for the only possible F. Hence $Q(F)$ is always isomorphic to $Hom_F(Types, Terms)$ while $M \Rightarrow M$ is isomorphic to $M \to M$ iff M is extensional as a model of untyped λ-calculus. This example clearly shows the links between $\Rightarrow$ and $\to$. It is easily seen that trivial models cannot be complete: they obviously satisfy Axiom C, because $Types$ is a singleton and all polymorphic maps are constant.*

2. *Let us illustrate Point 2 of the definition of polymax models with the simplest models of [7]. In this case $Terms = (P(\Omega), \subseteq)$, where Ω is some fixed set and $P(\ )$ denotes the full power set operator, and $Types = (\{ X_D \,/\, D \in P(\Omega) \}, \subseteq)$, where $X_D = (P(D), \subseteq)$. The situation is similar, even if more refined, for all webbed models of F (cf. Appendix C).*

   Note that each element of $Terms$ will in general belong to infinitely many elements of $Types$ (this is true in fact for all the known non trivial polymax models), This is true in particular for the definable elements of $Terms$. Thus, even though we take into account the type information when we interpret the explicitly typed term t^σ in a polymax model, the interpretation of t^σ is not only in the interpretation of σ, but also in many other "semantic types" in the sense of Point 2.

3. *In universal retraction models $X \Rightarrow Y$ is the range of a retraction, and is directly isomorphic to $X \to Y$ (with no intermediate encoding via traces), and similarly with Q. In particular such a model is extensional even if it is obtained from a universal retraction on a nonextensional model of untyped λ-calculus.* [17]

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

5. *The PER-models satisfy the first 5 conditions of the definition of (extensional) polymax models, except for the convenient but not essential condition requiring that the carrier sets of objects of $Univ$ are non empty, but they do not satisfy $6 + 7$ since it is easy to produce in these models two F, F' such that $Q(F) = Q(F')$ and $Hom_F(Types, Terms)$ is different from $Hom_{F'}(Types, Terms)$, contradicting Remark 6. This is detailed in Appendix B.2.*

[17] Strictly speaking, this remark makes sense only for the universal retraction models which fit the present definition of polymax (otherwise $X \Rightarrow Y$ makes no sense). But a similar remark could be made in the wider setting of [7], and hence for all universal retraction models when described as polymax models as in Section 7.2 of [7].

In the rest of the paper we will use the following simplified notations.

Notation 8 *("semantic applications")*
From now on all conceivable notions of application will be denoted along the pattern $\psi(\xi)$. This means precisely that we adopt the following abbreviations:

$f(x)$ *is usual and*	$\in Y$	*if*	$f \in Hom(X,Y)$,	$x \in X$
$f(x) := appl_{X,Y}(f)(x)$	$\in Y$	*if*	$f \in X \Rightarrow Y$,	$x \in X$
$f(X)$ *is usual and*	$\in Terms$	*if*	$f \in Hom(Types, Terms)$,	$X \in Types$
$f(X) := Appl(f)(X)$	$\in Terms$	*if*	$f \in Terms$,	$X \in Types$
$F(X)$ *is usual and*	$\in Types$	*if*	$F \in Hom(Types, Types)$,	$X \in Types$

All previous semantic applications have a syntactic counterpart in F. In the completeness proof we will also need to be able to apply a type to a type (see Section 6). This will be done by requiring later on the existence of a morphism $AP \in Hom(Types, Types \to Types)$ (in fact: of a left-inverse for Q).

Notation 9 *("semantic applications" continued) Assuming the existence of a left-inverse AP for Q for the first and second row (the third one is just a reminder).*

$F(X) := AP(F)(X)$	$\in Types$	*if*	$F \in Types$ *and* $X \in Types$
$\forall X.F(X) := Q(AP(F))$	$\in Types$	*if*	$F \in Types$
$\forall X.F(X) := Q(F)$	$\in Types$	*if*	$F \in Hom(Types, Types)$

Notation 10 *("semantic abstractions"):*

$\lambda x^X.f(x^X)$	*means*	$lbd_{X,Y}(f)$	*if*	$f \in Hom(X,Y)$
$\lambda X.f(X)$	*means*	$Lambda(f)$	*if*	$f \in Hom(Types, Terms)$

In particular $\lambda x^X f(x^X)$ makes sense even if X doesn't interpret a type.

This notation is compatible with the preceding one in the sense that :
$(\lambda x^X.f(x^X))(u) = f(u)$ for all $u \in X$
$(\lambda X.f(X))(Z) = f(Z)$ for all $Z \in Types$

Definition 11 *The elements of $Terms$ of the form $\lambda X.f(X)$ will be called* polymorphic maps. *This is a very generic term which covers in particular the interpretations of polymorphic terms.*

Notation 12 *(Arrows and Currifications).*
As usual $A, B \to C$ abbreviates $A \to (B \to C)$ and $A^2 \to C$ abbreviates $A, A \to C$. We adopt the similar abbreviation for " $\Rightarrow$ ". Also, following a standard notation for "$\to$", we will let $\forall X.A \Rightarrow B$ abbreviate $\forall X.(A \Rightarrow B)$.
We will use Currification freely. Moreover, in order to increase readability, we will use Currification-like abbreviations for $\Rightarrow$ and for polymorphism, extending them even to lists of arguments mixing elements of $Types$ and elements of semantic types (i.e. elements of elements of $Types$). By this we mean, for example, that for $j \in \forall X.\forall Y.(X \Rightarrow Y)$ and $x \in X$, we will write $j(X,Y,x)$ or

$j(X,Y)(x)$ instead of $j(X)(Y)(x)$. Similarly, for $h \in \forall X.\forall Y.(X, Y \Rightarrow F(X,Y))$ and $x \in X$, $y \in Y$ we will write $h(X,Y)(x,y)$ instead of $h(X)(Y)(x)(y)$. In Appendix D, where no ambiguity can occur, we will even use abbreviations like $hXYxy$.

Finally, we will often use $x : X$ for $x \in X \in Types$.

5 The completeness result.

We now isolate a set $\mathcal{C}$ of conditions on polymax models, which are true in the BB-model and more generally in infinitely many webbed models (where they can be easily checked), and which are sufficient to imply completeness. The main result of this paper is indeed:

Theorem 13 *Every extensional polymax model $\mathcal{M}$ satisfying $\mathcal{C}$ is $\beta\eta$-complete, in the sense that for all F-types σ, τ and for all F-terms t, u having the same type:*
(i) we have $\sigma =_\alpha \tau$ if and only if, for all assignments ρ, $\mathcal{M} \models \rho(\sigma) = \rho(\tau)$.
(ii) we have $t =_{\alpha\beta\eta} u$ if and only if, for all assignments ρ, $\mathcal{M} \models \rho(t) = \rho(u)$.

In this section we describe the conditions and discuss their possible satisfaction in the various kinds of models.

5.1 Stating and discussing the conditions.

We divide the set of conditions $\mathcal{C}$ 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. A last and more artificial condition states that $Types$ contains exactly one representative of the terminal object of $Univ$ and will allow us to deal with it. These conditions are easy to satisfy, and, indeed, are 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 natural numbers. The reader can think here of O being the flat domain of integers and N as the set of its non-bottom elements.[18].
2. There are morphisms on O representing the successor and the predecessor function on N.
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 $f(Y(X)(f)) = Y(X)(f)$.

[18]This will be the case in the BB-model.

4. For all objects K of $Univ$ there is a fixed point operator $Y_K \in (K \to K) \to K$ and there is a conditional operator $IF_K \in O, O, K, K \to K$, such that for all $k, k' \in N$,
   $IF_K(k, k', a, b) = a$ if $k = k'$ and b otherwise.
5. *("Triviality conditions")*
   a) There is a unique element E of $Types$ which is terminal for $Univ$; we will call it "*the trivial type*".
   b) For all $X \in Types$ we have $(X \Rightarrow E) = E$.
   c) $Q(T) = E$, where $T \in Types \to Types$ is the constant morphism which takes value E on all types. We will refer to T as *"the trivial constructor"*.

Remark 14 About recursive and pairing functions. *It is a clear consequence of Conditions I. 1,2,4 that all the partial recursive functions on N will be representable in the model. In particular there will be a morphism $< -, - > \in Hom(O \times O, O)$ with left inverse (p_1, p_2), and such that: if $x, y \in N$, then $< x, y > \in N$ and $< x, y >$ is strictly bigger than x, y (for the usual ordering of integers). We will make extensive use of this pairing function in the sequel. Requiring the existence of such a pairing morphism in Cond. I.2, instead of a successor and a predecessor morphisms, would be sufficient for proving the completeness theorem, but there would be a (slight) cost* [19].

Remark 15 About the triviality conditions $I.5$. *They are somewhat artificial but very mild, and they are unavoidable in the following sense. First all the concrete models which are polymax in the sense of the present paper* [20] *are such that $Types$ contains one and only one (representative of "the") terminal object. In these models we have furthermore $E = \bot_{Types}$, so that introducing E allows not only for technical simplifications later on, but also for a better visualization of the models we are interested in. Furthermore all these models satisfy the other triviality conditions, for the following reasons. Since $Univ$ is a category of domains, both $E \in Types$ and $X \to E$ (which is not in $Types$) are singleton domains. Thus the equation $(X \Rightarrow E) = E$ is satisfied as soon as $X \Rightarrow E$ is isomorphic to $X \to E$. In particular, this is true for all the extensional models (and hence for all the universal retraction models). The equation $X \Rightarrow E = E$ is also true for all the models of [7], whether extensional or not, due to the specific definition of $\Rightarrow$ for them. The situation with Q is similar.*

[19] The cost is that in order to prove Corollary 31 it would be necessary to admit terms $\lambda\alpha.t$ where *α is not free in the type of a free variable of $nf(t)$* (instead of t). This leads us to work with an extension F' of F which is "inessential", in some sense, and has in particular the "same" term model than F with respect to (an extended notion of) $\alpha\beta\eta$-equivalence. In particular a few trivial remarks about this extension are enough to conclude that the proof which is given in the present paper works, with the only change that we have to consider equivalence classes of F'-terms instead of F-terms.

[20] The universal retraction models of McCracken and Berardi are such that $Types$ (which is not anymore a domain of domains but only a family of domains) contains infinitely many representative of the terminal object, since all constant maps are admissible retractions in these two cases.

These properties of E have the undesirable consequence (for us) that E is simultaneously in the range of Q and $\Rightarrow$, since $E = Q(T) = (E \Rightarrow E)$. Making them part of the completeness conditions, and under this precise statement, greatly simplifies life later on [21].

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 $\Rightarrow$ and 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 $Types$ (for a well chosen ρ). 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, namely an infinite family of semantic types $C(i)$ such 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 $\breve{f} \in \forall X.[O \Rightarrow C(index(X))]$, $\breve{g} \in \forall X.[C(index(X)) \Rightarrow O]$ such that $\breve{f}(C(i)), \breve{g}(C(i))$ 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 $P_1, P_2 \in Hom(Types, Types)$ such that (P_1, P_2) is a "quasi-inverse" for $\Rightarrow$. This precisely means that:
   for all $X, Y \in Types$, we have $P_2(X \Rightarrow Y) = Y$,
   if $Y \neq E$, we have furthermore $P_1(X \Rightarrow Y) = X$.
3. There is a morphism $AP \in Hom(Types, Types \to Types)$ reversing Q. In other words : for all "type constructors" $F \in Types \to Types$ we have $AP(Q(F)) = F$.
4. We have a case map $case \in \forall G.\forall X.(G(X)^3 \Rightarrow G(X))$ distinguishing among arrow types, quantified types, and "canonical values for variables". By this we mean that for all $G \in Types$, and $a, b, c \in \forall X.G(X)$ we have :
   $case(G)(X)(a(X), b(X), c(X)) = a(X)$ if $X = C(i)$.
   $case(G)(X)(a(X), b(X), c(X)) = b(X)$ if $X = Y \Rightarrow Z$ and $X \neq E$.
   $case(G)(X)(a(X), b(X), c(X)) = c(X)$ if $X = Q(F)$ and $X \neq E$.

[21] In order to have a more natural statement, a referee suggested to work more generally with the set $Triv$ of all the possible representatives of the terminal object in$Types$. However the proof of (the analogue of) Lemma 21 below would then need two problematic closure properties of $Triv$. On the contrary, the present Lemma 21 is here easily proved, since the problematic properties become trivially true when $Triv$ is a singleton.

5. There is a "trace operator" between any two types, that is, there 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 j is that $j(X,X,x) = x$ for all $X \in Types$. We make no assumption over the value of $j(X,Y,x)$ in any other case (it will then often be $\perp_Y$).

Remark 16 *We could not ask for a total left-invertibility of $\Rightarrow$ in condition $II.2$, since $X \Rightarrow E$ is trivial for all X ($I.5$). Similarly we were obliged to exclude E from the conditions imposed on the polymorphic map "case", since E is simultaneously in the range of $\Rightarrow$ and of Q. The conditions on the case function imply however that E is the only such element.*

Remark 17 *We observe in [8] that any polymax model which is such that Q is left-invertible is in fact an implicit model of Girard's F_ω. The strong reason why our completeness proof cannot be extended to F_ω is that no (even more accurate) polymorphic map case could ever be able to discriminate between the ranges of (the interpretation of) the various higher-order quantifiers, since, in these models, all these ranges are included in the range of Q.*

Remark 18 *It is immediate from $I.5$ and $II.2$ that $(X \Rightarrow Y) = E$ iff $Y = E$. Similarly it is immediate from $I.6$ and $II.3$ that $Q(F) = E$ iff $F = T$.*

We will see now that the technical restrictions linked to E, even though unavoidable, are inessential in the sense that the previous remark implies that we can manage to avoid meeting E during the completeness proof, by using only strict environments (Lemma 21).

Definition 19 *A type-environment $\rho : Vartypes \to Types$ is* strict *if it only takes nontrivial values (on type variables).*

Example 20 *(The "canonical environment"). The key example of a strict environment is the environment ρ_C defined by $\rho_C(\alpha_i) = C_i$ for all i.*

Lemma 21 *For all F-types σ and all strict environments ρ, we have $\rho(\sigma) \neq E$.*

Proof. By induction on σ, using Remark 18. The only nonobvious case is $\sigma := \forall\alpha.\tau$. Then $|\sigma|_\rho := Q(X \mapsto |\tau|_{\rho[\alpha:X]})$. Let $K : Types \to Types$ be the argument of Q. Then for any nontrivial type Y the environment $\rho[\alpha : Y]$ is strict and we have $|\tau|_{\rho[\alpha:Y]} \neq E$ by the inductive hypothesis; hence K is far from being trivial (i.e. equal to T [22]) and hence $Q(K) \neq E$. ■

5.2 Satisfaction of the conditions in concrete models.

Theorem 22 *The BB-model is extensional and satisfies the two groups of conditions. More generally this is true of infinitely many webbed models.*

[22] T was defined in Condition $I.5$.

For a proof of the first assertion we refer to [5]. The second one, which makes the BB-model part of a more general context, follows from easy observations which are made below and in [7] (see also Appendix C). For a reader familiar with [7] it should be clear that the construction of the BB-model can furthermore be varied so as to obtain complete models which satisfy a great variety of other constraints.

Remark 23 *The conditions of group II are easy to satisfy. On the other hand, it is also very easy* not *to satisfy them. For example.*

- *It is easy to see that models of FC cannot satisfy $II.4$.*
- *There is another kind of model which obviously cannot satisfy $II.4$: namely those models for which $range(\Rightarrow) \cap range(Q) \neq \{E\}$, including those models for which Q is onto. It is easy to build such webbed models.*
- *In contrast, it is easy* for a polymax model which contains infinitely many types called C_i *to satisfy $II.4$. Roughly speaking, it is enough that $Types$ can be decomposed into three independent "factors" which contain respectively: the range of $\Rightarrow$, the range of Q, and all the C_i. Such domain (in)equations can easily be forced in the framework of webbed models (see Appendix C)*
- *Stable models (those living in categories where morphisms are stable functions) cannot satisfy $II.2$, unless they are trivial. Indeed it is easy to check that P_1 cannot be stable if $II.2$ is true and $Types$ is not a singleton.*

Example 24

- *1. All the webbed models of [7] satisfy Conditions $II.3$ and $II.5$. It is very easy to characterize those which satisfy $II.2$, and very easy to build infinitely many webbed models which satisfy furthermore: the existence of suitable C_i 's and Condition $II.4$ (using the remark above).*
- *2. Trivial models obviously satisfy $II.2, 3, 5$, and do not satisfy $II.1, 4$.*
- *3. All universal retraction models satisfy $II.5$ but those that belong to the stable semantics do not satisfy $II.2$. We do not know whether universal retraction 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)* [23]*. It does not satisfy II.4 since it is a model of FC, and it does not satisfy $II.5$ since $\forall X.(\forall Y.X \Rightarrow Y)$ is a trivial type there.*
- *6. The situation with the continuous CGW model is similar for $II.3$ and $II.5$, but the argument given for $II.4$ doesn't apply since the continuous CGW is not a model of FC.*

[23] The analogue of our Q is not injective in Girard's model since, for example, both $\forall\alpha.\alpha$ and $\forall\alpha.\forall\beta.\alpha \to \beta$ take value $\{\emptyset\}$ in it (see [18] for the first type).

6 Definition by case and by recursion.

In this section we show how the conditions allow us to define new polymorphic maps (by case and) by recursion. As a matter of fact the (short) proof of Lemma 26 below uses all the conditions of group II, plus Condition I.3 (existence of fixed point operators Y_X over all $X \in Types$). Moreover the lemma even encapsulates, in some sense, the conditions of group II since, except for $II.1$ which we use only partially below, the other conditions II will not be used elsewhere. Lemma 26 will allow us to build the crucial polymorphic map *type* in Section 9 and the family (f, g) in Section 10.

Lemma 25 *(Definition by cases). let* $F : Types \to Types$ *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* $X \neq E$ *and* $X = Y \Rightarrow Z$ *for some* Y, Z.
3. $f(X) = c(X) : F(X)$ *if* $X \neq E$ *and* $X = Q(K)$ *for some* K.

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

Before stating the next lemma we recall that the retraction pair $(C, index) : O \rightleftarrows Types$ was introduced in Condition II.1 of Section 5.

Lemma 26 *(Definition by recursion). Let* $F : Types \to Types$ *and let :*

$a : \forall Z.F(C(index(Z)))$,
$b : \forall Y.\forall Z.(F(Y), F(Z) \Rightarrow F(Y \Rightarrow Z))$,
$c : \forall G.[\forall Z.F(G(Z))] \Rightarrow F(\forall Z.G(Z)))$.

Then there is some $f : \forall X.F(X)$ *such that, for all* $i \in N$, *for all* $Y, Z \in Types$ *and* $K \in Types \to Types$ *such that* $Z \neq E$ *and* $K \neq T$ *we have:*

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

Recall that T is the trivial constructor (which takes constant value E).

Note that the typing of c in the hypothesis implicitly uses the morphism AP, to define $G(Z)$.

Proof. We may translate the claim 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 $\forall X.F(X)$.
We define $H(f) : \forall X.F(X)$ as follows.
First we set:

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

Then we use the case definition lemma (Lemma 25) in order to define the body $H' := H(f)$ of H. In this definition we need to use the embedding j for forcing a', b', c' to get the type required by the definition by cases.

$H'(X) = j(F(C(index(X))), F(X), a'(X))$ $: F(X)$ if $X = C_i$
$H'(X) = j(F(P_1(X) \Rightarrow P_2(X)), F(X), b'(X))$ $: F(X)$ if $X = Y \Rightarrow Z$ and $Z \neq E$
$H'(X) = j(F(Q(AP(X))), F(X), c'(X))$ $: F(X)$ if $X = Q(K)$ and $K \neq T$

Now,

$C(index(X)) = X$ 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 $X \neq E$
$Q(AP(X)) = X$ if $X = Q(K)$ for some K.

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

$H'(C_i) = a(C_i)$ $: F(C_i)$
$H'(Y \Rightarrow Z) = b(Y, Z)(f(Y), f(Z))$ $: F(Y \Rightarrow Z)$
$H'(Q(K)) = c(Q(K))(\lambda Z.f(K(Z)))$ $: F(Q(K))$

It is then clear that any fixed point of $H := \lambda f.H'$ has the required properties.

■

7 An overview of the completeness proof.

The aim of this section is to give a survey of the completeness proof, which will then be developed in the rest of the paper. We start from Friedman's proof [15] for simply typed extensional λ-calculus [24], and then we introduce progressively the variants we had to adopt and the specific tools that we need.

7.1 Friedman's proof.

In this section (only) T denotes the term model of simply typed calculus, T_{tp} denotes the set of its types, and T_σ the set of $\beta\eta$-equivalence classes of terms of type σ. Models stem from applicative structures $M = (M_\sigma, apl_{\sigma,\tau})_{\sigma,\tau \in T_{tp}}$, where M_σ is a set where to interpret the variables (and the terms) of type σ, and $apl_{\sigma,\tau}$ is a map from $M_{\sigma\to\tau} \times M_\sigma$ to M_τ, submitted to an extensionality condition. The *full type structure* over the set B is the structure defined by: $M_\alpha = B$ for all type variable (or constant) α and $M_{\sigma\to\tau}$ is the set of all functions from M_σ to M_τ, while $appl_{\sigma,\tau}$ is the usual application of a function to its argument. The aim of Friedman is to show that this model is complete if B is infinite.

The core idea of Friedman's proof is the following:

1. In order to prove that a model M of the simply typed extensional λ-calculus is complete it is enough (see point 3. below) that there exists a partial homomorphism $(g_\sigma)_{\sigma \in T_{tp}}$ from M to T. In this context "homomorphism" refers to the applicative structure, and to be *a partial surjective homomorphism* means that each g_σ is a partial surjective map from M_σ to T_σ, and that for all σ, τ we have $g_{\sigma\to\tau}(x)$ defined as the unique element of T_τ, if any, such that for all $y \in Dom(g_\sigma)$ we have $g_{\sigma\to\tau}(x).g_\sigma(y) = g_\tau(apl_{\sigma,\tau}(x)(y))$.

[24] We have failed, up to now, to adapt to F the much simpler and stronger Statman-Simpson argument.

2. Supposing there is a family of partial surjective maps g_α from M_α to T_α, for any variable α, and supposing that for all σ, τ any partial surjective function from M_σ to M_τ is represented in $M_{\sigma\to\tau}$, there is a partial homomorphism from M to T, which can be built by induction on the structure of σ. In the case of the full type structure over B the second condition is fulfilled (since all functions are allowed) and the first one is equivalent to "B is infinite".

3. The completeness of M is an immediate consequence of a lemma which states that: given any partial homorphism $(g_\sigma)_{\sigma\in T_{tp}}$ there is a semantical environment ρ such that, for all types σ and all terms $t : \sigma$, we have $g_\sigma(\rho(t)) = t$. The lemma follows from a generalized version which also quantifies over syntactic interpretations π preserving types, and replaces ρ by an environment ρ_π built from π using the axiom of choice, in order to force the lemma at the level of type variables.

7.2 Overview of our proof.

When trying to extend this proof to adequate models of F the difficulties we met and the solutions we brought, are the following.

1. From Reynolds [30] we know that for no model of System F we will have that, for all σ, τ, the full function space is represented in $M_{\sigma\to\tau}$. In particular there is no reason why a function known only from its applicative behavior should be represented in $M_{\sigma\to\tau}$. The way to get off is to notice that in Friedman's case the existence of a partial applicative homomorphism is equivalent to the following statement (using the axiom of choice for finding f_σ).

   "There exists a family of pairs of functions $(f_\sigma, g_\sigma)_{\sigma\in T}$ between T_σ and M_σ such that such that $g_\sigma \circ f_\sigma = id_{T_\sigma}$ and, for all types σ, τ, for all $x \in M_{\sigma\to\tau}$ and $z \in T_\tau$, we have $g_{\sigma\to\tau}(x).z = g_\tau(apl_{\sigma,\tau}(x)(f_\sigma(z)))$".

   (There is no harm in the context of the full model to assume that g is total since we can choose arbitrary values outside the range of f_σ).

   The natural generalization to a context allowing less freedom is to ask for a family of retraction pairs (f_σ, g_σ), between T and M, which should be furthermore applicative homomorphisms in the sense above. Since we are dealing with System F, we should also add a similar condition for dealing with polymorphic types. We are now faced with the following questions.

   a) in which category will our retraction pairs live? If all the M_σ are objects of a category $Univ$ one may try to encode the term model within an object O of $Univ$. Then it makes sense to require that (f_σ, g_σ) is a pair of morphisms of $Univ$. Asking for a retraction pair between O and M_σ would be too strong (O is a priori bigger than the term model, and not yet stratified via the σ 's); this will lead us to define later on a notion of *pseudo retraction pair* with adequate properties.

b) the applicative condition suggests how to build g_σ inductively, but not f_σ (which should be built simultaneously); so it should be rephrased and refined; we should also state a second-order version, for dealing with $(f_{\forall\alpha.\sigma}, g_{\forall\alpha.\sigma})$.

c) anyway, we will not be able to define directly the pairs (f_σ, g_σ) by direct induction on syntactic types, because of the impredicativity of F. The classical way to get through, which dates back to Girard [17], is to use auxiliary objects, called "candidates" which range over a set whose definition is not inductive. If we work with a polymax model $\mathcal{M}$, the obvious choice is (the carrier set of) $Types$, and we will look for pairs $(f_X, g_X)_{X \in Types}$. For proving their key properties we will need later on much more complex candidates, inspired by Plotkin's logical relations and which will be here relations "wrapping" pseudo retraction pairs.

2. The fact that there are no analogues T_X of the T_σ 's in the term model is a second good reason to look for a richer structure and to encode the term model in it. Condition $I.1$ allows us to encode types and terms in O via morphisms (the code of t will be denoted by $\#t$). To encode the term model we need furthermore to introduce two (external) partial equivalence relations $\sim_{tp}$ and $\sim_{tr}$ (tp for "types" and tr for "terms") such that two elements of O are equivalent if and only if they encode α-equivalent types (resp. $\alpha\beta\eta$-equivalent terms). We let $\breve{O}_\sigma$ be the set of all the codes of terms of type σ in O and $\breve{O}_{tp}$ be the set of all the codes of types.

3. We are now able to define an adequate notion of *pseudo retraction pair*. Let D be a subset of O closed under $\sim$, which is either $\sim_{tp}$ or $\sim_{tr}$, and A an object of $Univ$. A pair of morphisms $(\varphi, \psi) : O \rightleftarrows A$ is a *pseudo retraction pair between D and A* if, for all $x \in D$ we have $\psi(\varphi(x)) \sim x$.

4. We are now looking for pairs $(f_X, g_X) : O \rightleftarrows X$ of morphisms, such that: 1. $g_{X \Rightarrow Y}$ and $g_{\forall X.F(X)}$ satisfy applicative conditions analogue to Friedman's one, but relative to adequate semantic notions of application (in O and in $\mathcal{M}$) and, 2. There is an interpretation ρ of type and term variables in $\mathcal{M}$, such that: a) for all type σ $(f_{\rho(\sigma)}, g_{\rho(\sigma)})$ is a pseudo retraction pair between $\breve{O}_\sigma$ and $\rho(\sigma)$, b) for all F-terms $t : \sigma$ we have $g_{\rho(\sigma)}(\rho(t)) \sim_{tr} \#t$.

   If we succeed, then $\beta\eta$-completeness of $\mathcal{M}$ follows immediately.

   We have to make precise here that ρ will act on types as the "canonical assignment" ρ_C defined from $\rho_C(\alpha_i) = C(i)$, where C is the morphism defined in Condition II.1.

5. First we need to be able to relate, as closely as possible and within $\mathcal{M}$, the set $\breve{O}_{tp}$, which represents the set of syntactic types in O, to $Types$. We prove the existence of a pseudo retraction pair $(type, [-])$ between $\breve{O}_{tp}$ and $Types$, which is furthermore such that $[\#\sigma] = \rho_C(\sigma)$. In particular $type(\rho_C(\sigma)) \sim_{tp} \#\sigma$ for all type σ, and hence any two non-α-equivalent

types will have different interpretations in $\mathcal{M}$. It is equivalent to build *type* : $\forall X.O$ with the same properties, and this can be done using the recursion lemma proved in Section 6. The construction of $[-]$ uses a more traditional recursion principle. The existence and properties of *type* and $[-]$, which are derived in Section 9, would fail in most models and strongly use the two groups of conditions that we assumed.

6. The morphisms (f_X, g_X) are built by recursion in Section 10; to be more accurate we build two polymorphic maps $f' : \forall X.O \Rightarrow X$ and $g' : \forall X.X \Rightarrow O$, with the right applicative behavior at induction steps, and we let $f_X := apl_{O,X}(f'X)$ and $g_X := apl_{X,O}(g'X)$. The base case of the recursion is obtained by setting $f'(C_i) = \breve{f}(C_i)$ and $g'(C_i) = \breve{g}(C_i))$, where the $\breve{f}, \breve{g}$ are the polymorphic maps given by Condition II.1. It is worth noting here that, to be able to define the family (f, g), we will need more than the applicative behavior of $f_{X\Rightarrow Y}, g_{X\Rightarrow Y}$, $f_{\forall X.F(X)}$, $g_{\forall X.F(X)}$: we will have to abstract over this behavior, at each step.

   In particular we are really forced to work with models of second order λ-calculus and not only with second-order applicative structures.

7. There remains to prove that the family also satisfies the conditions 2.a and 2.b of Point 4, which is done in Section 11. For $\sigma = \alpha_i$ this follows from the properties of $\breve{f}, \breve{g}$ (Condition II.1). The other cases are proved by induction on σ, using candidates which are less rigid than pseudo retraction pairs, but which have pseudo retraction pairs as backbones. First we associate to each pseudo retraction pair (φ, ψ) a set of relations which we call (φ, ψ)-*relations*. Finally, the candidates, which we call *sharp candidates*, are defined as the pairs (X, R) such that X is a nontrivial element of $Types$, $type(X)$ is the code of some type, say σ, and R is an (f_X, g_X)-relation between $\breve{O}_\sigma$ and X (Section 11.3). As mentioned in 1. this part of the proof is also in the spirit of Plotkin's logical relations [26] [27].

Some other technicalities have to be solved in the meanwhile: 1. From the natural quantification and abstraction first-order morphisms on O which come naturally when encoding the term model we must build higher-order [25] morphisms, which are better behaved in the sense that they form pseudo retraction pairs with the diverse applications (end of Section 8.3). 2. We need to work with syntactic interpretations which do not preserve types. 3. We have to be careful at some points when dealing with α-equivalence, since α-equivalent types and terms are not confused in O.

8 Type and terms encodings.

In this section we show how to encode the term model of System F into O, in such a way that it makes sense with respect to the whole model $\mathcal{M}$, and will

[25] in the sense that the arguments are no longer only in O but also in $O \to O$.

allow us later on to build the meaningful correspondence between O and $\mathcal{M}$ that we will need for proving the completeness of $\mathcal{M}$.

For sake of simplicity, we keep the same notation for an integer and its representation in O. The encodings will be done using a fixed but arbitrary left-invertible morphism $< -, - >\in Hom(O \times O, O)$, such that for all $x, y \in N$ we have $< x, y >\in N$ and $< x, y >$ strictly bigger than x, y for the usual order of integers. The existence of such a pairing morphism follows from the first Conditions I (Remark 14).

8.1 Types and terms encodings.

First we fix explicit (and arbitrary) enumerations of type and term variables: α_i for type variables, and x_i^σ for term variables of each type σ. Then we fix the following (rather arbitrary) encoding of F-types and F-terms in O.

For types we let:

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

For terms we let:

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

This encoding is an injective map from the set of types and terms into $N \subseteq O$. The general definition of the morphisms $var, arrow$, etc., from $\langle -, -\rangle$, can be inferred easily from their action on codes above.

Notation 27 *The conventions linked to Currification allow us to write $Ap1(v)$ for the morphism $w \mapsto Ap1(v, w)$, and similarly for $Ap2$.*

Notation 28 *We denote by $\breve{O}_{tp}$ the set of all codes of F-types. We denote by $\breve{O}$ the subset of N which is the set of all codes of F-terms, by $\breve{O}_\sigma$ the set of codes of F-terms of type σ, and by $\breve{O}_\forall$ the union of all those $\breve{O}_\sigma$ such that σ begins with $\forall$. Because of the typing rules of F, two α-equivalent σ, σ' give the same $\breve{O}_\sigma$.*

8.2 Interpreting the term model in O.

We first introduce two (external) equivalence relations over O which will allow us to view the term model when looking at O.

Notation 29 *$x \sim_{tp} y$ if $x = y$ or x and y encode two α-equivalent types.*
$x \sim_{tr} y$ if $x = y$ or x and y encode two $\alpha\beta\eta$-equivalent F-terms.
We use a generic name "$\sim$" when more convenient.

Let us make precise a technical point, which is always omitted when working only up to α-equivalence but which plays a role at the level of encodings. By definition, two variables x_i^σ and $x_{i'}^{\sigma'}$ are α-equivalent iff $i = i'$ and σ and σ' are α-equivalent; by $t[x^\sigma : x_i^\sigma]$ we understand that all the free occurrences of all $x^{\sigma'}$, σ' α-equivalent to σ, are replaced by x_i^σ.

By definition of $\sim$, all the encoding morphisms are compatible with $\sim_{tp}$ and $\sim_{tr}$ in the sense that they induce operations between the T_σ's and between T_σ and T_{tp}, where $T_\sigma := \breve{O}_\sigma / \sim_{tr}$ and $T_{tp} := \breve{O}_{tp} / \sim_{tp}$. In particular we really have an (encoding of) the term model of λ-calculus.

In the rest of the paper we will use directly the morphisms $v\alpha r, arrow, Ap1$, and $Var, Ap2$, which correspond to the second order applicative structure underlying the term model; in contrast $forall$, $Lb1$ and $Lb2$ will only be used through the higher-order morphisms called $quant$, $La1$ and $La2$, which we build in the next subsection. These morphisms will be used in the definition of the polymorphic maps $type$ and (f, g).

8.3 Reconstructing a type or term from its extensional behavior.

We show here that we can reconstruct via higher-order morphisms (the code of) a type or term, up to equivalence, from its extensional behavior (in the sense of its action on all possible variables). That these functions are morphisms will follow from the fact that they are obtained from $< -, - >$, p_1, p_2 (see Remark 14) and from the basic morphisms of the underlying *c.c.c.* (composition, evaluation, projections, etc.).

To begin with, the following lemma expresses in particular that, once an enumeration of type or term variables has been fixed, there is a *uniform* way to recover a type (resp. a term), up to α- (resp. $\alpha\beta\eta$-) equivalence from all its substitution instances (where a fixed variable is successively replaced by all the variables of the same kind in the enumeration). This uniformity will then allow us to find morphisms able to reconstruct uniformly the types and terms which begin by $\forall$ or λ from their applicative behavior (in the good cases).

Lemma 30 *Let e be a type (resp. a term). Let ξ be a term or a type variable, let ξ_i be the fixed enumeration of the term or type variable of the same kind as ξ* [26] *and finally suppose that $e[\xi : \xi_i]$ makes sense* [27]*. Then, for all $f : O \to O$ if we set $r :=< f(0), 0 >$ we have:*

$$\forall i \in N \; (\, f(i) \sim \#(e[\xi : \xi_i]) \,) \;\Rightarrow\; (\xi \equiv \xi_r \text{ or } \xi_r \text{ is not free in } nf(e)$$

Proof. Let $\simeq$ denotes α- (resp. $\alpha\beta\eta$-) equivalence. We recall that $nf(e)$ is the normal form of e if e is a term, and is e itself if e is a type.

[26] Thus, if ξ is a type variable, then $\xi_i := \alpha_i$. If ξ is a term variable of type σ, then $\xi_i := x_i^\sigma$.

[27] The only case where it would not make sense is when e is a type and ξ a term variable.

By hypothesis $f(0)$ is $\sim$-equivalent to the code of a type (resp. of a term), namely $\#(e[\xi : \xi_0])$. Hence $f(0)$ is itself the code of a type (resp. of a term), by definition of $\sim$. Let e'_f be such that $\#e'_f = f(0)$; still by definition of $\sim$ we have then: $e'_f \simeq e[\xi : \xi_0]$. Because of the hypothesis on the pairing function we have $\langle a, b\rangle > a, b$ for all a, b; therefore we have $\#\xi_r \geq r =< f(0), 0 > > \#e'_f$. By applying the same hypothesis several times, we deduce that ξ_r cannot appear in e'_f, otherwise it would follow $\#\xi_r < \#e'_f$. Thus ξ_r is not free in $nf(e[\xi : \xi_0])$ since ξ_r is not free in $nf(e'_f)$ and both are α-equivalent. This is equivalent to saying that either $\xi = \xi_r$ or ξ_r is not free in $nf(e)$. ■

Corollary 31 *1. (Reconstruction of terms). There is a morphism* $La1 : O, (O \to O) \to O$ *such that, for all types* σ*, all morphisms* $g : O \to O$*, and all* F*-terms* t*,*

$$\begin{array}{c} \textit{if } g(\#x_i^{\sigma'}) \sim_{tr} \#t[x^\sigma : x_i^\sigma] \textit{ for all } i \in N \textit{ and all } \sigma' \simeq_\alpha \sigma \\ \textit{then } \ La1(\#\sigma, g) \sim_{tr} \#(\lambda x^\sigma.t) \end{array}$$

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

$$\begin{array}{c} \textit{if } \ h(\#\alpha_i) \sim_{tr} \#t[\alpha : \alpha_i] \textit{ for all } i \in N, \\ \textit{then } \ La2(h) \sim_{tr} \#(\lambda\alpha.t) \end{array}$$

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

$$\begin{array}{c} \textit{if } \ k(i) \sim_{tp} \#(\tau[\alpha : \alpha_i]) \textit{ for all } i \in N, \\ \textit{then } \ quant(k) \sim_{tp} \#(\forall\alpha.\tau). \end{array}$$

Proof. As in the previous proof, $\simeq$ denotes α- (resp. $\alpha\beta\eta$-) equivalence.

We only treat the second case; the other two ones are similar, and in fact they are simpler (see remark below). Applying Lemma 30 to $f := h \circ var$, $\xi := \alpha$, and $\xi_i := \alpha_i$, we get that $\alpha_r \equiv \alpha$ or α_r is not free in $nf(t)$. In both cases $\lambda\alpha.t \simeq \lambda\alpha.nf(t) \simeq \lambda\alpha_r.nf(t)[\alpha : \alpha_r] \simeq \lambda\alpha_r.nf(t[\alpha : \alpha_r]) \simeq \lambda\alpha_r.nf(t_f)$, where t_f is such that $\#t_f = f(r)$, since $t_f \simeq t[\alpha : \alpha_r]$ and hence $nf(t_f)$ is α-equivalent to $nf(t[\alpha : \alpha_r])$. Note that in both cases all these terms are well-formed terms (either because α is not free in the type of a free variable of t or because α_r is not free in $nf(t)$). Thus $\#\lambda\alpha.t \sim_{tr} \#\lambda\alpha_r.nf(t_f)$.

Now, since the model is able to represent all the recursive functions on N, there is a morphism $k : O \to O$ which computes the code of the normal form of any term from the code of this term. Now we have: $\#(\lambda\alpha.t) \sim_{tr} \#(\lambda\alpha_r.nf(t_f)) = Lb2(var(r), k(\#t_f)) = Lb2(var(r), k(f(r)))$. Hence: $\#(\lambda\alpha.t) \sim_{tr} La2(h)$ with $La2(h) := Lb2(\, var(< f(0), 0 >)\, ,\, k(f(< f(0), 0 >))\,)$.

Then $La2$ is a morphism, since $f := h \circ var$ and since $Lb2$ is a morphism. ■

The following remarks will only be used in the proof of Lemma 54.

Remark 32 *(Compatibility of* $La1$ *with respect to* $\sim$*).*
If $\#\sigma \sim_{tp} \#\sigma'$*, then* $La1(\#\sigma, g) \sim_{tr} La1(\#\sigma', g)$.

Remark 33 *(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 $La1(\#\sigma, g) \sim_{tr} La1(\#\sigma, g')$, $La2(h) \sim_{tr} La2(h')$ and $quant(k) \sim_{tp} quant(k')$.

Remark 34 *1. $La1(\#\sigma, Ap1(v)) \sim_{tr} v$ for all types σ and all $v \in \breve{O}_{\sigma\to\tau}$.*

This follows from the definition of $La1$, applied to $g := Ap1(v)$.

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: $La1(\#\sigma, Ap1(v)) \sim_{tr} \#(\lambda y^\sigma.ty^\sigma) \sim_{tr} \#t = v$.

2. $La2(Ap2(v)) \sim_{tr} v$ for all $v \in \breve{O}_\forall$.

The proof is similar and follows from the definition of $La2$ applied to $h := Ap2(v)$.

8.4 Pseudo retraction pairs.

The following notion of pseudo retraction pair is very natural in our context.

Definition 35 *Given $\emptyset \neq D \subseteq \breve{O}$ closed under $\sim$ (which is either $\sim_{tp}$ or $\sim_{tr}$), and given an object A of $Univ$, a* pseudo retraction pair (φ, ψ) between D and A *is a pair of morphisms $(\varphi, \psi) : O \rightleftarrows A$ such that, for all $x \in D$, we have $\psi(\varphi(x)) \sim x$. Finally we say that (φ, ψ)* is a pseudo retraction pair *if it is a pseudo retraction pair between some D, A.*

We have already met such pseudo retraction pairs, since the remarks at the end of the previous subsection can be rephrased as follows:
1. $(Ap2, La2)$ is a pseudo retraction pair between $\breve{O}_\forall$ and $O \to O$, and
2. for all type σ, $(Ap1, La1(\#\sigma))$ is a pseudo retraction pair between $\cup_\tau \breve{O}_{\sigma\to\tau}$ and $O \to O$.

But the key examples are yet to come. The first one is built in the next section.

9 Linking $\breve{O}_{tp}$ and $Types$: the pseudo retraction pair $([-], type)$.

In this section we will prove the existence of a pseudo retraction pair $([-], type)$ between $\breve{O}_{tp}$ and $Types$, which is moreover such that *type* computes the (integer code of) a type out of its canonical interpretation in $\mathcal{M}$. This pair will be crucial for proving that $\mathcal{M}$ is $\beta\eta$-complete; in particular, it will be used in the definition of the family (f, g).

The morphism *type* will in fact be built below as an element $\forall X.O$ [28], using the general recursion principle developed in Section 6, while the definition of $[-]$ will use a more traditional recursion principle based on Condition I.4.

[28] More accurately, the pseudo retraction pair will be $(Appl(type), [-])$. Since we will be interested only in the applicative behaviour of *type* there is no problem in confusing both.

Lemma 36 *There is an element type of $\forall X.O$ such that:*

1. $type(C_i) = var(i) = \#\alpha_i$
2. $type(Y \Rightarrow Z) = arrow(type(Y), type(Z))$ *for* $Z \neq E$.
3. $type(Q(K)) = quant(i \in O \mapsto type(K(C_i)))$ *for* $K \neq T$.

Proof. The existence of *type* follows immediately from Lemma 26 applied 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 37 *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, following rules recalled in the Section E). We will say that π* is compatible with *ρ if:*

1. *ρ is strict*
2. $\#(\pi(\alpha_i)) \sim_{tp} type(\rho(\alpha_i))$.

If π is compatible with ρ then $\pi(\alpha_i)$ is determined by $\rho(\alpha_i)$, up to α-equivalence. In the sequel we keep the same notation for the environments and their extensions to all types. The following example is the key one.

Example 38 *Let ρ_C be the strict 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.*

Remark 39 *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 environments are unchanged on $\alpha_j \neq \alpha$, and since $\rho[\alpha : \alpha_i](\alpha) = C_i$ and $\pi[\alpha : \alpha_i](\alpha) = \alpha_i$ (and $type(C_i) = \#\alpha_i$).*

Lemma 40 *If π is compatible with ρ then, for all σ, $\rho(\sigma)$ is nontrivial and* $\#(\pi(\sigma)) \sim_{tp} type(\rho(\sigma))$.

Proof. The first assertion is given by Lemma 21, since ρ is strict. For the second one see Appendix F. ■

Lemma 41 *There is a morphism $[-] : O \to Types$ such that, for all $x \in \breve{O}_{tp}$ we have $type([x]) \sim_{tp} x$.*

This statement is a corollary of the more general lemma:

Lemma 42 *There is a morphism $[-]_- : O, (O \to Types) \to Types$ such that, for all $r : O \to Types$ and for all types σ, we have: $[\#\sigma]_r = \rho_r(\sigma)$, where ρ_r is the environment defined by $\rho_r(\alpha_i) := r(i)$*

Proof. (of Lemma 41 from Lemma 42).
We set $[-] := [-]_C$, where C is the morphism introduced by Condition $II.1$. By Lemma 42, if we restrict x to $\breve{O}_{tp}$, we have $[x] = \rho_C(\#^{-1}(x))$. Now, we have $type(\rho_C(\#^{-1}(x))) \sim_{tp} x$, by Lemma 40 applied to $\pi = id$ and $\rho = \rho_C$; hence $type([x]) \sim_{tp} x$. ■

Proof. (of Lemma 42).
The definition of $[-]_-$ is by recursion and by cases (Cond. I.4), using the left invertibility of $\langle -, - \rangle$ (Remark 14); more precisely, from I.4, one uses the existence of IF_{Types} and $Y_{O,(O \to Types) \to Types}$; see Appendix F for the details. ■

In the sequel the pair $(type, [-])$ is taken as any fixed pair of morphisms satisfying the conditions stated in Lemma 36 and Lemma 41.

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

We will build the family $(f_X, g_X) : O \rightleftarrows X$ in three steps.

In a first step we present the most natural way, in our setting, to obtain:
1. pairs $(f_{X \Rightarrow Y}, g_{X \Rightarrow Y})$ out of two pairs (f_X, g_X) and (f_Y, g_Y)
2. pairs $(f_{Q(K)}, g_{Q(K)})$ out of all the $(f_{K(X)}, g_{K(X)})$
just requiring that they have the right domains and ranges.
The aim is to use this information as the starting point of a definition by recursion of the family (f, g). However such a definition is only possible for polymorphic f, g's and $\forall X.(O \to X)$ makes no sense. Thus, we will rather use recursion to build a pair (f', g'), such that $f' \in \forall X.(O \Rightarrow X)$, and $g' \in \forall X.(X \Rightarrow O)$ and such that the pairs (f_X, g_X) defined by $f_X := apl_{O,X}(f'(X))$ and $g_X := apl_{X,O}(g'(X))$ satisfy the constraints above. The behavior of f', g' on C_i will be fixed according to Cond. II.1.

In a second step we will translate these constraints on (f, g) into recursion conditions on (f', g').

In the third step. We will check that Lemma 26 really applies, in other words: that the expressions prescribing the behavior of f', g' on the range of $\Rightarrow$ and of Q have the shape of adequate polymorphic maps b, c.

Of course, for general X, we will have little information on (f_X, g_X), and in models based on domains they will often return the value $\perp$. We will however be able to show later on that this pair is a pseudo retraction pair whenever X is the interpretation of some F-type in a suitable environment (Lemma 54).

10.1 Intuitive definition of (f, g) at induction steps.

In this preliminary step we present the most natural way to get:
1. pairs $(f_{X \Rightarrow Y}, g_{X \Rightarrow Y}) : O \rightleftarrows (X \Rightarrow Y)$ out of two pairs $(f_X, g_X) : O \rightleftarrows X$ and $(f_Y, g_Y) : O \rightleftarrows Y$
2. pairs $(f_{Q(K)}, g_{Q(K)}) : O \rightleftarrows Q(K)$ out of all the $(f_{K(X)}, g_{K(X)}) : O \rightleftarrows K(X)$, for $K \in Types \to Types$

For $\Rightarrow$ we first note that, from $(f_X, g_X) : O \rightleftarrows X$ and $(f_Y, g_Y) : O \rightleftarrows Y$ we can define a pair $(\phi_{X,Y}, \psi_{X,Y}) : (O \to O) \rightleftarrows (X \to Y)$, by $\phi_{X,Y}(h) := f_Y \circ h \circ g_X$ and $\psi_{X,Y}(h') := g_Y \circ h' \circ f_X$.

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 (componentwise) the pair $(Ap1, La1(type(X)) : O \rightleftarrows (O \to O)$ with the pair $(\phi_{X,Y}, \psi_{X,Y}) : (O \to O) \rightleftarrows (X \to Y)$, and then with the retraction pair $(lbd_{X,Y}, apl_{X,Y})$:

$$(f_{X\Rightarrow Y}, g_{X\Rightarrow Y}) := (lbd_{X,Y}, apl_{X,Y}) \circ (\phi_{X,Y}, \psi_{X,Y}) \circ (Ap1, La1(type(X)))$$

Hence, $(f_{X\Rightarrow Y}, g_{X\Rightarrow Y})$ is defined from (f_X, g_X) and (f_Y, g_Y) by:

$$f_{X\Rightarrow Y}(x) = lbd_{X,Y}(f_Y \circ Ap1(x) \circ g_X) \qquad \text{for all } x \in O \tag{1}$$

$$g_{X\Rightarrow Y}(y) = La1(type(X),\ g_Y \circ apl_{X,Y}(y) \circ f_X) \quad \text{for all } y \in X \Rightarrow Y \tag{2}$$

For Q (or $\forall$) we need the morphisms *type* and $[-]$ (defined in Section 9).

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

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

We can note that the equations (1) and (2) are mutually recursive, while (3) and (4) do not mix up f and g.

10.2 The recursive constraints on (f', g').

For $f' : \forall X.(O \Rightarrow X)$ and $g' : \forall X.(X \Rightarrow O)$ to be such that $f_X := apl_{O,X}(f'_X)$ and $g_X := apl_{X,O}(g'_X)$ fulfill the equations above it is enough (see below) that f' and g' should satisfy the following conditions, which are written using the abbreviations introduced at the end of Section 4 (otherwise $Lambda, Appl$, and various instances of lbd, apl would appear).

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

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

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

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

The values of f', g' on the $C(i)$'s are given by Cond. II.1: $f'(C_i) = \breve{f}(C_i)$ and $g'(C_i) := \breve{g}(C_i)$.

Remark 43 *For all $x \in O$ we have: $f'_X(x) = f_X(x)$ since both are equal, by definition, to $(apl_{O,X}(f'_X))(x)$. Similarly: $g'_X(u) = g_X(u)$ for all $u \in X$.*

It is straightforward to check that, conversely, the constraints on f', g' above are enough to imply the correct behavior of the morphisms f_X, g_X. Indeed it is only a matter of replacing the "application and abstraction abbreviations" used in the expressions of f', g' above by their real values.

For example, let us check that (5) implies (1) (the other cases are similar).

For all $x \in O$ we have:

$f_{X\Rightarrow Y}(x) = f'_{X\Rightarrow Y}(x)$ (Remark above)

$= \lambda z^X.\ f'_Y(\ Ap1(x)\,(g'_X(z))\ )$ (by (5))

$= \lambda z^X.\ f'_Y(\ Ap1(x)\,(g_X(z))\ )$ (Remark above)

$= lbd_{X,Y}(z \in X \mapsto f'_Y(\ Ap1(x)\,(g_X(z))\ )$ (by definition of $\lambda z^X.t$ for $t \in Y$)

$= lbd_{X,Y}(f_Y \circ Ap1(x) \circ g_X)$, as required.

10.3 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 will follow from the recursion Lemma 26 applied to $F(X) := (O \Rightarrow X) \times (X \Rightarrow O)$. Here $U \times V := \forall Z.(U, V \Rightarrow Z) \Rightarrow Z$ is the interpretation in the model of the product defined in System F. The reason why we are forced to work with pairs (at the formal level) is of course that some of the conditions on f', g' are mutually recursive.

We have to show that there exist three polymorphic maps a, b, c satisfying the hypothesis of the recursion lemma (Lemma 26). We fix map a by *Cond* II.1, and have to extract map b from (5),(6) and c from (7),(8).

To convince oneself of the existence of b it is enough to notice that the equations which compute $f'(X \Rightarrow Y)$ and $g'(X \Rightarrow Y)$ in terms of X, Y, $f'(X)$, $f'(Y)$, $g'(X)$, $g'(Y)$ are correct at the level of (semantic) types, and to check that they combine these data using only types and morphisms available in the model.[29] In particular, O (at the type level), and $type, Ap1, La1$ (at the term level), if we only mention those parameters whose existence follows from the completeness conditions. This is similar for c, except that we use the parameters O, AP, $type, [-], Ap2, La2$.

Extracting b, c is, theoretically, just a straightforward rewriting exercise. In practice it is rather awful for the following reasons: first, because we are obliged to deal only with pairs, without even being able to rely on some symmetry between f and g (since there is none, if we except their domains and codomains); and, second, because of the presence of explicit typing. Since, we will only use the recursive equations in the sequel, we can postpone the explicit extraction of b, c to Appendix D, where it is easy to check that they have the required form.

From now on, (f_X, g_X) always refers to the family defined in this section. For the sake of readability, we will not mention the pairs (lbd, apl) and $(Lambda, Appl)$ anymore, and will use "$\circ$" to denote all possible compositions in the model. In particular, there will be no visual distinction between the pairs (f, g) and (f', g') and we will simply write :

$$
\begin{array}{ll}
f_{X \Rightarrow Y}(x) = & f_Y \circ Ap1(x) \circ g_X \quad (\text{for } x \in O), \\
g_{X \Rightarrow Y}(y) = & La1(type(X),\ g_Y \circ y \circ f_X) \quad (\text{for } y \in X \Rightarrow Y), \\
f_{Q(K)}(x) = & \lambda X.\ f_{K(X)}(Ap2(x, type(X))) \quad (\text{for } x \in O), \\
g_{Q(K)}(y) = & La2(r \in O \mapsto\ g_{K([r])}(y([r]))) \quad (\text{for } y \in Q(K)).
\end{array}
$$

11 The heart of the completeness proof.

11.1 The main result.

The $\beta\eta$ completeness of $\mathcal{M}$ is an immediate consequence of the following:

[29] Note that this is true only because $\mathcal{M}$ is polymax.

Proposition 44 *There is an environment ρ on type and term variables, such that, for all type σ, and for all F-terms $t : \sigma$ we have $g_{\rho(\sigma)}(\rho(t)) \sim_{tr} \#t$.*

Indeed, let t, u be any closed terms of type σ; If t, u are equated in the model, then, in particular, $\rho(t) = \rho(u)$ for the environment ρ mentioned above, and $\#t \sim_{tr} \#u$. Thus $t =_{\beta\eta} u$, which was our claim.

Since we cannot prove the Proposition directly by induction over σ, and since furthermore reasoning directly with the (f_X, g_X) does not seem possible, we will rather reason about kinds of logical relations which admit skeletons which are pseudo retraction pairs.

The rest of the section is devoted to the proof of the Proposition.

11.2 (φ, ψ)-relations.

We are interested in relations which have, in some sense, a skeleton which is a pair of adequate morphisms.

Definition 45 *Given any pair $(\varphi, \psi) : O \rightleftarrows X$ of morphisms, and given $D \subseteq \breve{O}$ which is closed under $\sim_{tr}$ and nonempty, we call* (φ, ψ)-relation between D and X, *any $R \subseteq D \times X$ such that :*

1. *$x \in D$ implies $R(x, \varphi(x))$.*
2. *$R(x, y)$ implies $x \sim_{tr} \psi(y)$.*
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.

Remark 46 *(i) If R is a (φ, ψ)-relation between D and X, then (φ, ψ) is a pseudo-retraction pair between D and X* [30].

(ii) Conversely, given a pseudo retraction pair (φ, ψ) between D and X there is a minimum *(φ, ψ)-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 $y = \varphi(x'))$.*

(iii) D is not uniquely determined from R and (φ, ψ) : if $D' \subseteq D$ is closed under $\sim_{tr}$, then (φ, ψ) is still a pseudo retraction pair between D' and X and R is a (φ, ψ)-relation between D' and X.

Remark 47 *Let us here motivate the use of general (φ, ψ)-relations instead of the minimum ones (which would amount to work directly with the pseudo-retraction pairs (φ, ψ)). We observe that, given an (φ, ψ)-relation R, the set $R(x) := \{y \,/\, R(x, y)\}$, which only depends of the $\sim_{tr}$-equivalence class of x, lies between $\varphi(\{x' \,/\, x' \sim_{tr} x\})$ and $\psi^{-1}(x' \,/\, x' \sim_{tr} x)$. This gives us crucial flexibility when handling the (φ, ψ)-relations, while working only with the minimum ones would tie us to the lower bound and would prevent desired closure properties of the set of relations.*

[30] pseudo retraction pairs were defined in Section 8.4.

11.3 Candidates.

In this section all interpretations are relative to types, in particular π and ρ always denote a syntactic and a semantic environment, in the sense we already met.

Definition 48 *A* sharp candidate is a pair (X, T) *such that* $X \in Types$, X *is non trivial,* $type(X) \in \breve{O}_{tp}$, *and* T *is an* (f_X, g_X)*-relation between* $\breve{O}_{\#^{-1}(type(X))}$ *and* X.

Definition 49 *A* sharp 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 binary relations. Then a *sharp relational environment* can be defined as a pair $(\rho, \mathcal{R})$, where ρ is an environment on types and $\mathcal{R}$ is 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 $\breve{O}_{\pi(\alpha_i)} \times \rho(\alpha_i)$, where π is the unique environment on types compatible with ρ.
Note that uniqueness of π is here only up to α-equivalence, as in Definition 37, and recall that $\breve{O}_\sigma$ is, for all σ, independent of the α-equivalence class of σ.

In particular each sharp relational environment Υ induces a pair (ρ, π).

Definition 50 *The environments* Υ, ρ and π are compatible *if* ρ *and* π *are induced by* Υ *in the sense above, and are furthermore compatible in the sense of Definition 37.*

Example 51 *Let* Υ_C *be the sharp relational environment defined by* $\Upsilon(\alpha_i) := (C_i, R_i)$ *where* R_i *is the minimum* (f_{C_i}, g_{C_i})*-relation between* $\breve{O}_{\alpha_i}$ *and* C_i. *Then* Υ_C, ρ_C *and id are clearly compatible, because of Condition II.1.*

We now present the formulas which will allow us to interpret all types by sharp candidates. Recall that we already know how to extend the ρ-components to all types: $\rho(\sigma \to \tau) := \rho(\sigma) \Rightarrow \rho(\tau)$ and $\rho(\forall\alpha.\sigma) := Q(X \mapsto \rho[\alpha : X](\sigma))$ (Appendix A). We will use the following remark.

Remark 52 *If* Υ *is compatible with* (ρ, π), *then for any candidate* (X, T), *we have that* $\Upsilon[\alpha : (X, T)]$ *is a sharp relational environment, and that* $\Upsilon[\alpha : (X, T)]$ *is compatible with* $\rho[\alpha : X]$ *and* $\pi[\alpha : \#^{-1}(type(X)]$.

Definition 53 *We define* $\Upsilon(\tau) := (\rho(\tau), \mathcal{R}^\tau_\Upsilon)$, *where* ρ *is the environment induced by* Υ *and* $\mathcal{R}^\tau_\Upsilon$ *is a binary relation on* $O \times \rho(\tau)$, *which only depends on the value of* Υ *on the free variables of* τ. *The definition is by induction on* τ, *and at each step for all environments* Υ. *We let:*

R1. $\mathcal{R}^{\alpha_i}_\Upsilon$ *is the second component of* $\Upsilon(\alpha_i)$.

R2. $\mathcal{R}^{\sigma\to\tau}_\Upsilon(x, y)$ *iff* $y \in \rho(\sigma \to \tau)$ *and* $\forall v, w\ (\mathcal{R}^\sigma_\Upsilon(v, w) \Rightarrow \mathcal{R}^\tau_\Upsilon(Ap1(x, v),\ y(w)))$.

R3. $\mathcal{R}^{\forall\alpha\tau}_\Upsilon(x, y)$ *iff* $y \in \rho(\forall\alpha.\tau)$ *and for all candidates* (X, T), *we have :*
$\mathcal{R}^\tau_{\Upsilon[\alpha:(X,T)]}(Ap2(x, type(X)),\ y(X))$.

Note that the relations do not discriminate between objects which have the same applicative behavior, and hence between the interpretations of η-convertible terms. This is the point which prevents us from dealing with β-completeness as well.

Lemma 54 *For all τ, and for all Υ, ρ, π which are compatible, $\mathcal{R}^\tau_\Upsilon$ is an $(f_{\rho(\tau)}, g_{\rho(\tau)})$-relation between $\breve{O}_{\pi(\tau)}$ and $\rho(\tau)$, hence $(\rho(\tau), \mathcal{R}^\tau_\Upsilon)$ is a sharp candidate.*
In particular $(f_{\rho(\tau)}, g_{\rho(\tau)})$ is a pseudo retraction pair between $\breve{O}_{tp}$ and $\rho(\tau)$.

Proof. The proof, which is detailed in Appendix F, goes by induction on τ. The induction steps use that $Ap1$, $Ap2$, La_1 and La_2 are compatible with $\sim$, and more generally uses all the machinery developed in Section 8.3. ■

11.4 Linking $\pi(t)$ and $\rho(t)$.

¿From now on:

ρ denotes any interpretation of *F-terms and F-types* in $\mathcal{M}$, the environment from which it is issued, and its restriction to the world of types. It should be clear from the context which one we are really using. By the definition of such interpretations we necessarily have $\rho(t) \in \rho(\sigma)$ if $t : \sigma$ (it is convenient here to use ρ instead of $|-|_\rho$).

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

Υ is a relational interpretation on types, as defined in Section 11.3.

Definition 55 *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 (in the sense of Definition 37)*
2. *$\mathcal{R}^\sigma_\Upsilon(\,\#\pi(x^\sigma_i), \rho(x^\sigma_i)\,)$ 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 $\pi(x^\sigma_i)$ is determined, up to $\alpha\beta\eta$-equivalence, by $\rho(x^\sigma_i)$ and the restrictions of Υ to the type variables which are free in σ. The following result is a key one.

Lemma 56 *There exist Υ, ρ such that Υ links id to ρ, where id is the identity on types and terms.*

Proof. Let $\Upsilon := \Upsilon_C$ and let ρ be the extension of the canonical type environment ρ_C by $\rho(x^\sigma_i) = f_{\rho(\sigma)}(\#x^\sigma_i)$. It is immediate that Υ links id and ρ (we already know that Υ,ρ and id are compatible at the level of types (Example 51)). ■

We have already seen that the compatibility conditions between Υ,ρ and π propagate to all types (Lemma 40 and 54). The next proposition states that they propagate to all terms.

Proposition 57 *If Υ links π to ρ, then for all τ, we have :*
$\mathcal{R}^\tau_\Upsilon(\,\#\pi(t), \rho(t)\,)$ for all F-terms $t : \tau$.

Proof. By induction on the length of t, and at each step for all strongly compatible Υ, ρ, π. The proof is detailed in Appendix F. ■

Corollary 58 *For all types τ, and all terms $t : \tau$ we have $g_{\rho_C(\tau)}(\rho_C(t)) \sim_{tr} \#t$.*

Proof. From Proposition 57 and the fact that Υ_C links ρ_C to *id*, we have $\mathcal{R}^{\tau}_{\Upsilon_C}(\,\#t, \rho_C(t)\,)$ for all types τ and all terms $t : \tau$. From Lemma 54 we know that $\mathcal{R}^{\tau}_{\Upsilon_C}$ is an $(f_{\rho_C(\tau)}, g_{\rho_C(\tau)})$-relation between $\breve{O}_\tau$ and $\rho_C(\tau)$. Thus we have $g_{\rho_C(\tau)}(\rho_C(t)) \sim_{tr} \#t$, as required. ■

Proposition 44 follows immediately, which ends the completeness proof.

A Interpretations of types and terms in polymax models.

Once our models have been presented, the interpretation of types and terms below is very simple. Furthermore the fact that the interpretation is correct follows from a lemma whose statement is simple and whose proof is both straightforward and simple (see [7, Section 2]). However, the truth of this last sentence relies strongly on the fact that we admit as many polymorphic functions as possible. Otherwise the justification lemma would not only be more difficult to state, but its proof would also raise genuine problems (which, for example, forbid us to develop a stable analogue of the class of models built in [7]).[31]

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 4.

Interpretation of types.

Definition 59 *A* type-environment *is a function ρ from $Vartypes$ to $Types$.*

Given a type environment ρ and $X \in Types$, we denote by $\rho[\alpha : X]$ the environment ρ' defined by : $\rho'(\alpha) := X$, and ρ' coincides with ρ everywhere else.
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.

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

For the well-definedness of $|-|_\rho$ we refer to [7].

[31] A similar problem already arises in Girard's (stable) interpretation of F in [18]. In this case the interpretation finally works because of very strong properties of Girard's model, but a conceptual and nontrivial argument has to be developed for dealing with the case of an application of a term to a term (contrary to the assertion in [18, Section 3.10]); this seems to be well known now.

Interpretation of terms.

Definition 60 *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 variables x^σ we have $\rho(x^\sigma) \in |\sigma|_\rho$ (hence $\rho(x^\sigma) \in Terms$).*

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 $\rho'(x^\sigma) := v$. When σ is not considered up to α-equivalence, we have to require that the value of $\rho(x^\sigma)$ is independent of the class of σ, and to change the definition of $\rho[x^\sigma : v]$ accordingly.

It is not possible to define an environment $\rho[\alpha : X]$, essentially because there is no canonical choice then for $\rho[\alpha : X](x^\sigma)$ if α is free in σ. We are however able to give a local meaning to the expression $|t|_{\rho[\alpha:X]}$.

The interpretation of F-terms under all possible environments is by induction on the complexity of the term t :

- $|x^\sigma|_\rho := \rho(x^\sigma)$
- $|t^{\sigma\to\tau} u^\sigma|_\rho := apl_{|\sigma|_\rho,|\tau|_\rho}(|t^{\sigma\to\tau}|_\rho)\,(|u^\sigma|_\rho)$
- $|\lambda x^\sigma.t^\tau|_\rho := lbd_{|\sigma|_\rho,|\tau|_\rho}(\, v \in |\sigma|_\rho \mapsto |t^\tau|_{\rho[x^\sigma:v]}\,)$.
- $\left|t^{\forall\alpha.\sigma}\tau\right|_\rho := Appl(\left|t^{\forall\alpha.\sigma}\right|_\rho)\,(|\tau|_\rho)$
- $|\lambda\alpha.t^\sigma|_\rho := Lambda\,(\, X \in Types \mapsto |t^\sigma|_{\rho[\alpha:X]}\,)$.

Using the abbreviations introduced at the end of Section 4, this can be written simply as:

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

For the correctness of this definition we refer to [7].

B Connections with other settings.

B.1 Polymax models induce Bruce-Meyer-Mitchell models.

There is a rough, uniform, and straightforward way to view polymax models as particular models in the sense of Bruce-Meyer-Mitchell [11], which we will make explicit below. But it is worth noting beforehand that for some subclasses of polymax models we can provide much sharper translations for the kinds and for their related "maps" Φ_{κ_1,κ_2} (in the notation of [11]). First example: as soon as Q is left-invertible (which is true for all our webbed models) we can get models of F_ω just by giving a more accurate interpretations of the Φ_{κ_1,κ_2}'s. Second example: for the simplest "square webbed models of F", and in particular for $\mathcal{E}^2$, we can provide a still sharper interpretation in which kinds are interpreted by elements of $Types$, and which gives a model of λ^* [8]. In the following, BMM stands for the Bruce-Meyer-Mitchell framework.

The following table shows how the polymax models can be viewed as second order frames, in the sense of [11]. The fact that they are furthermore environment models follow from the (rather straightforward) proof that the interpretation of F in polymax models is correct (now in [7]).

The left-hand side of the table concerns the BMM syntax and notion of models, and we keep the BMM notation, while the right-hand side concerns polymax models, and uses our notations. In particular, on the right-hand side, the arrow $\to$ denotes the exponent operator of the *ccc* $Univ$.

In the table the expression "*is interpreted by*" relates a syntactic construct and its semantic interpretation in a polymax model, while "*is translated by*" only means a change of notation. The other expressions are, we hope, non ambiguous.

Note that the object $Terms$ has no analogue in the BMM framework, where it is not required that all terms be interpreted in the same "domain".

BMM		*Polymax*
$C_{term} =_{def} \emptyset$		
$C_{cst} =_{def} \{\Rightarrow, \forall\}$		
T	is interpreted by	$Types$
$\Rightarrow$ (arrow over kinds)	is interpreted by	$\rightarrow$ (the exponent of $Univ$)
$\rightarrow$ (arrow over types)	is interpreted by	$\Rightarrow$ (the morphism on $Types^2$)
$Kind^{\kappa_1}$	is chosen as	the interpretation $\|\kappa_1\|$ of κ_1
$Kind^T$	is hence equal to	$Types$
$Kind^{\kappa_1 \Rightarrow \kappa_2}$	is hence equal to	$\|\kappa_1\| \rightarrow \|\kappa_2\|$
$[Kind^{\kappa_1} \rightarrow Kind^{\kappa_2}]$	is chosen as	$Hom(\|\kappa_1\|, \|\kappa_2\|)$
Φ_{κ_1,κ_2}	will associate	$Hom(\|\kappa_1\|, \|\kappa_2\|)$ to $\|\kappa_1\| \rightarrow \|\kappa_2\|$ (and hence will take its carrier set)
$a \in Kind^T$	is translated by	$X \in Types$
Dom^a	is translated by	*carrier set of* (X)
$Dom^{a \rightarrow b}$	is hence translated by	*carrier set of* $(X \Rightarrow Y)$
$\Phi_{a,b}$	is chosen as	$apl_{X,Y}$
$f \in Kind^{[T \Rightarrow T]}$	is translated by	$F \in Hom(Types, Types)$
$\forall f$	is chosen as	$Q(F)$
$\prod_{a \in Kind^T} Dom^{f(a)}$	has its role played by	$Hom_F(Types, Terms)$
Φ_f	is then chosen as	$Appl_F$

B.2 The PER-model is not polymax.

As mentioned in Example 7, the PER-models, when presented in the right setting, satisfy the first 5 conditions of the definition of polymax model, but do not satisfy Conditions 6+7. We hope that this counter-example will contribute to a clearer understanding of what polymax models really are. We develop in detail the case of the standard PER-model over ω, basing our presentation on [21], but the argument works obviously for all realizability models. The standard PER-model over ω uses Kleene's notion of application on ω, which will be denoted below by a dot.

$Univ$ is here the category of ω-sets: the objects of $Univ$ are the pairs $A \equiv (S, \Vdash_S)$ where S is a set and $\Vdash_S \subseteq \omega \times S$ is a surjective relation (called the *realizability* relation).

It is easy to check that, in this setting, $A \equiv (S, \Vdash_S)$ is a substructure of $A' \equiv (S', \Vdash_{S'})$ if and only if $S \subseteq S'$ and $\Vdash_S = (\omega \times S) \cap \Vdash_{S'}$. Hence any subset S of S' determines a unique substructure of $(S', \Vdash_{S'})$.

For $A \equiv (S, \Vdash_S)$ and $A' \equiv (S', \Vdash_{S'})$ one defines: $Hom(A, A')$ as the set of all realizable functions from S to S', where $n \in \omega$ realizes f if for all $x \in S$ and $m \in \omega$ we have $m \Vdash_S x$ implies $n \cdot m \Vdash_{S'} f(x)$. We denote the latter realizability relation by $\Vdash_{A \rightarrow A'}$ and define: $A \rightarrow A'$ as being $(Hom(A, A'), \Vdash_{A \rightarrow A'})$.

We let *types* [32] be the (mere) set whose elements are the partial equivalence relations, or *pers*, on ω, and we denote these pers by R, R'. We let $Eq(R)$ be the set of all the (nonempty) subsets of ω which are equivalence classes for R, and, if nRn, we let $n^R \in Eq(R)$ be the R-equivalence class of $n \in \omega$.

$Terms$ is defined as follows: the carrier set of $Terms$ is the set:

$\{ (x, R) / \ R \in types \text{ and } x \in Eq(R) \}$,

and we take $n \Vdash_{Terms} (x, R)$ iff $n \in x$.

For $R \in types$ we let X_R be the ω-set which is the substructure of $Terms$ whose carrier set is $\{ (x, R) / x \in Eq(R) \}$. In particular $X_\emptyset = \emptyset$ and $R \neq R'$ implies $X_R \cap X_{R'} = \emptyset$.

$Types$ is defined as follows: the elements of $Types$ are the X_R, $R \in types$, and we have $n \Vdash_{Types} X$ for all $n \in \omega$ and $X \in Types$.

We let T denote the bijection $R \mapsto X_R$ between *types* and $Types$.

Since the realizability relation is trivial on $Types$, we have that, for any object A of $Univ$, the set $Hom(A, Types)$ is the full function set between the carrier sets of A and $Types$. (*)

In particular $Hom(Types, Types)$ is the set of all endo-functions of $Types$.

We will denote $F \mapsto F^T$ the bijection induced by T between the endofunctions of $Types$ and those of *types*; more precisely: $F^T =_{def} T^{-1} \circ F \circ T$.

Definition of $\Rightarrow$ and Q. We first define two operations $\Rightarrow$ and q relative to the set *types*: for all endo-function G on *types* we define $q(G) =_{def} \cap G(R)$, while $\Rightarrow$ is the usual canonical way to associate to the pers R, R' a per R'' having the property that $X_{R''}$ is canonically isomorphic to $X_R \to X_{R'}$.

We then define $X_R \Rightarrow X_{R'}$ as $X_{R \Rightarrow R'}$ .Thus, for all R, R' we have an isomorphism pair between $X_R \to X_{R'}$ and $X_R \Rightarrow X_{R'}$.

Then, for all $F \in Hom(Types, Types)$, we define $Q(F)$ as $X_{q(F^T)}$.

We notice that $\Rightarrow$ and Q are morphisms, because of the remark (*) above.

With these definitions we clearly have that the PER-model satisfies the first 5 conditions of the definition of a polymax model (except for the non emptyness of the objects of $Univ$).

Note that in the PER-model Q is not injective, and that $Q(F)$ and $Q(F')$ are here equal or disjoint substructures of $Terms$. This contrasts the situation of the universal retraction models and the webbed models, for which Q is injective and $Q(F)$ and $Q(F')$ need not be disjoint if $F \neq F'$.

Furthermore it is easy to check that $Hom_F(Types, Terms)$ (or Hom_F for short) consists only of "constant" functions. More precisely $f \in Hom_F$ iff it has the right domain and range and there exists $n \in \omega$ such that for all $m \in \omega$

[32] The notation "*types*" fits the general notations of [7].

and $X_R \in Types$ we have $f(X_R) = ((n \cdot m)^{F^T(R)}, F^T(R))$; thus f is "constant" in the sense that $n \cdot m$ is independent of R.

Let us recall that Conditions 6+7 implies that $Q(F) \cap Q(F') = \emptyset$ implies $Hom_F \cap Hom_{F'} = \emptyset$, and that $Q(F) = Q(F')$ implies $Hom_F = Hom_{F'}$ (Remark 6).

To refute this last property in the PER-model we fix a proper non empty subset W of *types* and we let R_1, $R_2 \in types$ be two distinct pers such that $R_1 \cap R_2 \neq \emptyset$. We then let F be defined by $F^T(R) = R_1$ if $R \in W$ and $F^T(R) = R_2$ otherwise, and we let F' be the function on $Types$ obtained when exchanging the roles of R_1 and R_2. Then Hom_F and $Hom_{F'}$ are two nonempty disjoint sets, since $R_1 \neq R_2$, although $Q(F) = Q(F') = R_1 \cap R_2$.

Remark 61 *The PER-model could be presented more simply, taking $X_R =_{def} Eq(R)$, and $Terms =_{def} \bigcup_{R \in types} X_R$, but the presentation above is also pertinent for the discussion in Section B.3.2 below, while the simpler one is not.*

B.3 Polymax models vs categorical frameworks.

B.3.1 Internal models of F [3].

Internal models of F are parametrized by a cartesian closed category, say $Univ$, which is supposed to have all finite limits. Then the model consists essentially of a tuple $c \equiv (c_0, c_1, DOM, COD, COMP, ID)$, which is supposed to be an "internal Cartesian closed category", plus the existence of a relevant right adjoint, for ensuring a sound interpretation of syntactic types and terms. In particular c_0 and c_1 are objects of $Univ$ and the other constructs are morphisms. The role played by c_0 is exactly that of $Types$, while the object c_1 is intended to consist of all the internal morphisms between the elements of c_0. Intuitively the role played by the other constants is the following: DOM and $CODOM$ associate a domain and codomain in c_0 to each element of c_1, while $COMP$ allows the composition of elements of c_1, and finally ID associates to each element X of c_0 an element of c_1 which will play the role of the identity morphism on X.

Since in polymax models each $X \Rightarrow Y$ is a substructure of $Terms$, one could be tempted to make $Terms$ play the role of c_1. However, there is no way to define DOM and $CODOM$ on $Terms$, since an element of $Terms$ can be in no substructure of the form $X \Rightarrow Y$ or in infinitely many.

Basing the notion of model on c_1 rather than $Terms$ can also be viewed as natural, and is more economical, in some sense. However using it for studying particular models or classes of models is costly, and not only from a conceptual point of view: in practice the, a priori, very natural requirement on the existence of DOM and $CODOM$ may lead to unnecessary complicate encodings. For example the description of universal retraction models as polymax models in [7, Section 6.2] is much simpler and much more immediate than their description as internal models in [3, p. 278].

B.3.2 Pitts' topos models [25].

It was insistently suggested by one referee that the class of polymax models "was only a recast of Pitts' topos models of F". In fact polymax models and topos models are rather alien one to each other.

First, and using our notations, Pitts' definition, asks for the existence of a morphism $\tau \in Hom(Terms, Types)$ associating an element of $Types$ to each element of $Terms$ (in a way which would be relevant for the (later) intended interpretation of System F). The definition of polymax doesn't. Moreover, in all the known nontrivial polymax models there is no such morphism. Actually, in all these polymax models nearly all the elements of $Terms$, including the definable ones, enjoy infinitely many semantic types, in contrast to Pitts' models, and, furthermore, even when a concrete polymax model happens to own a canonical morphism from $Terms$ to $Types$ (which is, incidentally, the case for all the webbed models of [7]), then this correspondence is completely irrelevant to the interpretation of System F.

Conversely, the realizability models fit Pitts' framework, using the ω-set $Terms$ that we introduced in the previous section of this appendix, but are not polymax, as shown there.

B.3.3 Why not categorical polymax models?

As already mentioned, turning the definition of polymax models into a categorical one could only be done at the expense of a significant increase of complexity. From our point of view this could only prove useful when somebody will really need to include some concrete and "intuitively polymax" model not fitting the slightly more general definition that we gave in [7]. When trying to replace, in this latter definition, the notion of substructure by a more categorical one, the remark that, in all concrete polymax models, each $i_X : X \subseteq Terms$ is the half of an embedding-projection pair might be relevant. One should also replace $Hom_F(Types, Terms)$ by an object of $Univ$, but further work would be needed in order to have a coherent definition providing a class of sound models. It is unclear to us whether the resulting definition would really be significantly less complicated than the one in [3].

C Webbed models of System F.

The main purpose of this appendix is to help the reader to figure out what the webbed models look like, and why the BB-model fulfills conditions I and II. The reader is referred to [7] and [5] for more details and for real justifications. We follow the notations of [7].

All the models of [7] are generated from "F-webs", in a way similar to the way models of untyped λ-calculus can be generated from webs (e.g. Scott information systems). We define an F-web as a tuple

$$(\Omega, \Subset_{hom}, \Subset_{coh}, \preceq_{hom}, \preceq_{coh}, j_{hom}, j_{hcoh})$$

where Ω is a nonempty set, $\asymp_{hom}$ and $\asymp_{coh}$ are two binary, reflexive and symmetric relations (called respectively homogeneity and coherence), $\preceq_{hom}$ and $\preceq_{coh}$ are two preorders; finally j_{hom} and j_{hcoh} are injections from $\Omega_f^{hom} \times \Omega$ (resp. $\Omega_f^{hcoh} \times \Omega$) to Ω, where E_f^{hom} (resp. E_f^{hcoh}) is, for any subset E of Ω, the set of finite homogeneous (resp. finite homogeneous and coherent) subsets of Ω. The relations and functions are subject to compatibility conditions, which are described in [7].

Similarly as for untyped λ-calculus, the preorders are needed only mainly to force extensionality. For the simplicity of the exposition we will suppose below that they are trivial, and we will not mention them anymore.[33]

The compatibility conditions imply in particular that $(\Omega, \asymp_{hom}, j_{hom})$ generates a *reflexive* Scott domain, called *types* which, is in 1-1 correspondence with a "domain of domains" called $Types$. As a domain *types* is the set consisting of the homogeneous subsets D of Ω, ordered by inclusion. The elements of $Types$ are the domains X_D generated by the webs $(D, \asymp_{coh}^D)$, where $D \in types$ and $\asymp_{coh}^D$ is the restriction of $\asymp_{coh}$ to $D \times D$. The order on $Types$ is, by definition, the order induced by that of *types*, hence both domains are isomorphic via the correspondence $D \mapsto X_D$. There is also a domain $Terms$, which is generated by the web $(\Omega, \asymp_{hom} \cap \asymp_{coh})$. The role of the two injections j_{hom} and j_{hcoh} is to generate all the retraction pairs which are needed in the definition of a polymax model. All these retraction pairs encode and decode morphisms via their traces (in contrast to universal retraction models). The definitions of $\Rightarrow, Q$, $Lambda$ and $lbd_{X,Y}$ run as follows (they are defined relatively to *types*, and then can be trivially transferred to $Types$).

For $F \in types \to types$, $f \in types \to Terms$, $D, D' \in types$, $g : X_D \to X_{D'}$ we set:

$$\begin{array}{lcl} Q(F) & := & \left\{ j_{hom}(a,x) \;/\; (a,x) \in \Omega_f^{hom} \times \Omega \text{ and } x \in F(a) \right\} \\ Lambda(f) & := & \left\{ j_{hom}(a,x) \;/\; (a,x) \in \Omega_f^{hom} \times \Omega \text{ and } x \in f(a) \right\} \end{array}$$

$$\begin{array}{lcl} D \Rightarrow D' & := & \left\{ j_{hcoh}(a,x) \;/\; (a,x) \in D_f^{hcoh} \times D' \right\} \\ lbd_{D,D'}(g) & := & \left\{ j_{hcoh}(a,x) \;/\; (a,x) \in D_f^{hcoh} \times D' \text{ and } x \in g(a) \right\} \end{array}$$

As already mentioned when discussing the completeness conditions, all the webbed models satisfy conditions I.3 (existence of a uniform fixed point operator), I.5 (triviality conditions), II.3 (left-invertibility of Q) and II.5 (existence of the "trace" operator). Furthermore many of them satisfy II.2, which is the (quasi) left-invertibility of $\Rightarrow$. It is very easy to build webs which force the existence of an $O \in Types$ satisfying I.1,2,4, and in fact of infinitely many "independent" copies C_i of O satisfying II.1 (see below).

[33] The coherence and homogeneity relations can also be taken as trivial; such webs generate very simple *and non trivial* models of F.

It was also mentioned that to force the existence of the *case* function it was enough, knowing that all webbed models are polymax, to look for a webbed model where $Types$ could be seen as the coalesced sum $B \oplus C \oplus D$ of three independent factors B, C, D [34] containing respectively : the range of $\Rightarrow$, the range of Q, and finally all the C_i 's. In the next paragraph we show how this is achieved for the BB-model.

The F-web of (a non extensional version of) the BB-model is built by induction, starting from a "partial F-web" $(A, \asymp^A_{hom}, \asymp^A_{coh}, \emptyset, \emptyset)$, where: $A = N \times N$, where N is the set of nonnegative integers; the elements of A are pairwise incoherent, and they are homogeneous if and only if they have the same second component.

Then a definition by induction of $(\Omega, \asymp_{hom}, \asymp_{coh}, j_{hom}, j_{hcoh})$ is carried on in such a way that, at the end, the ranges of j_{hom} and j_{hcoh} are disjoints, are disjoints from A, and the compatibility conditions are satisfied. Thus we get an F-web which generates a domain $Types$ which can be decomposed as specified above. In this (version of) the model O and the C_i are isomorphic to the domain of flat integers, and are defined by $C_i := X_{N \times \{i\}}$ and $O = C_0$; in particular it is very easy to check Conditions $I.1$ and $II.1$.

When describing universal retraction models it is also convenient to use an auxiliary domain *types*. The elements of *types* are no longer webs but are retractions, and the elements of $Types$ are the ranges of these retractions (and are domains). The view of universal retraction models as polymax models is detailed in [7].

D The extraction of b, c.

For proving that the definition of (f', g') by recursion is correct we have to rewrite the recursive equations which define f', g' implicitely, so as to exhibit two polymorphic maps b, c satisfying the hypothesis of Lemma 26.

Step 1. Notations and abbreviations.
Recall $U \times V = \forall T.((U, V \Rightarrow T) \Rightarrow T)$
The interpretations of the pair construct and of the projections are:
$p := \lambda U.\lambda V.\, \lambda u^U \lambda v^V.\, \lambda T.\, \lambda z^{U,V \Rightarrow T}.\, zuv \ : \ \forall U.\forall V.\, (U, V \Rightarrow (U \times V))$
$\pi_1 := \lambda.U.\lambda V.\, \lambda z^{U \times V}.\, zU\, \lambda u^U.\lambda v^V.u \ : \ \forall U.\forall V.((U \times V) \Rightarrow U)$
$\pi_2 := \lambda U.\lambda V.\, \lambda z^{U \times V}.\, zV\, \lambda u^U.\lambda v^V.v \ : \ \forall U.\forall V.((U \times V) \Rightarrow V)$
To increase the readability we introduce the following (local) abbreviations which will allow us to provisionally forget some type indications.
For all $u \in U$, $v \in V$, $w \in U \times V$:
$< u, v >:= pUVuv \ : \ U \times V$
$\pi_1(w) := \pi_1 UVw \ : \ U$

[34] This means that $Types$ is the disjoint union of three domains B, C, D, except for their bottom element which are amalgamated.

$\pi_2(w) := \pi_2 UVw \ : \ V$
Finally we recall that $F := \forall X.((O \Rightarrow X) \times (X \Rightarrow O))$ and we set:
$\Phi := \lambda X. < f'_X, g'_X > := \lambda X.\, p(O \Rightarrow X)(X \Rightarrow O)\, f'(X)\, g'(X) \ : \ \forall X.F(X)$.
Thus, $f'_X = \pi_1(\Phi(X))$ and $g'_X = \pi_2(\Phi(X))$

Step 2. Extraction of b.
Now, it is clear that the pair of equations which defines $f'_{X\Rightarrow Y}$ and $g'_{X\Rightarrow Y}$ in terms of f'_X, f'_Y, g'_X and g'_Y, in Section 10.2, is equivalent to:
$\Phi(X \Rightarrow Y) =< \lambda x^O.\lambda z^X.\, \pi_1(\Phi(Y))(\, Ap1(x)\,((\pi_2(\Phi(X))(z)))\,),$
$\lambda y^{X\Rightarrow Y}.\, La1(\, type(X),\ r \in O \mapsto\ \pi_2(\Phi(Y))(y(\pi_1(\Phi(X))(r)))\,)\ >$
That is to say:
$\Phi(X \Rightarrow Y) =$ "b"$(X)(Y)\Phi(X)\Phi(Y)$ for:
"b" $:= \lambda X.\lambda Y.\, \lambda\xi^{F(X)}.\, \lambda\varsigma^{F(Y)}.$
$< \lambda x^O.\lambda z^X.\, \pi_1(\varsigma)(\, Ap1(x)\,(\pi_2(\xi)(z))\,),$
$\lambda y^{X\Rightarrow Y}.\, La1(\, type(X),\ r \in O \mapsto\ \pi_2(\xi)(y(\pi_1(\varsigma)(r)))\,)\ >$
and "b" : $\forall X.\forall Y.(F(X), F(Y) \Rightarrow F(X \Rightarrow Y))$.
To obtain the real value of b there remains to replace:
$\pi_i(\xi)$ by: $\pi_i(O \Rightarrow X)(X \Rightarrow O)(\xi)$,
$\pi_i(\varsigma)$ by: $\pi_i(O \Rightarrow Y)(Y \Rightarrow O)(\varsigma)$, and finally:
$< u, v >$ by $p\,(O \Rightarrow (X \Rightarrow Y))\,((X \Rightarrow Y) \Rightarrow O)\,(u)(v)$.

Step 3. Extraction of c.
The equations defining $f'_{Q(K)}$ and $g'_{Q(K)}$ in terms of $X \mapsto (f_{K(X)}, g_{K(X)})$ in Section 10.2 can be rewritten as:
$\Phi(Q(K)) =< \lambda x^O.\lambda X.\, \pi_1(\Phi(K(X)))(Ap2(x, type(X)))\,,$
$\lambda y^{Q(K)}.\, La2(\, r \in O \mapsto\ \pi_2(\Phi(K([r])))(y([r]))\,)\ >$
That is to say: $\Phi(Q(K)) =$ "c"$(Q(K))(\lambda Z.\Phi(K(Z)))$ for:
"c" $:= \lambda G.\lambda u^{\forall Z.F(G(Z))}.$
$< \lambda x^O.\lambda X.\, \pi_1(u(X))(Ap2(x, type(X)))\,,$
$\lambda y^G.\, La2(\, r \in O \mapsto\ \pi_2(u([r]))(y([r]))\,)\ >$
and "c" : $\forall G.[\forall Z.F(G(Z))] \Rightarrow F(\forall Z.G(Z)))$.
To obtain the real value of c there remains to replace:
$\pi_1(u(X))$ by: $\pi_1(O \Rightarrow G(X))\,((G(X)) \Rightarrow O)\,(u(X))$.
$\pi_2(u([r]))$ by: $\pi_1(O \Rightarrow G([r]))\,((G([r])) \Rightarrow O)\,(u([r]))$ and finally:
$< v, w >$ by $p\,(O \Rightarrow \forall Z.G(Z))\,((\forall Z.G(Z)) \Rightarrow O)\,(v)(w)$.

Remark 62 *It is then easy to check that b is the interpretation in the model of a closed polymorphic F-term with parameters in the model (including O, La_1, Ap1 and type), and that this is not true for c because of the presence of AP (which is hidden behind all the applications of G to Z, X and [r]). And this is already true at the level of types: the (semantic) type of c is not the interpretation of an F-type, while this is true for b (with parameter O).*

E Syntactic interpretations (or substitutions).

Syntactic interpretations associate types to types and F-terms to F-terms in such a way that $t : \sigma$ implies $\pi(t) : \pi(\sigma)$. They are defined from their restrictions to type and term variables (or environments) by the inductive rules given below; the definition is only up to α-equivalence. These interpretations preserve (α- and) $\alpha\beta\eta$-equivalence.

We denote by π the global interpretation, its restriction to types, and the environment 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, for $t : \sigma$, the substitution which behave like π on type variables and on all term variables different from x_i^σ, and gives value t to x_i^σ. If α is not free in the type of a free variable of t then $\pi[\alpha : \sigma](t)$ can be given a meaning analogous to the one we made precise in the case of semantic environments.

First we give the classical definition which considers types and terms only up to α-equivalence. Then we will be more refined in order to make our life easier in the proofs of Lemma 40, 54, and 57 in Appendix F.
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 $i \in N$,

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

$\pi(\forall\alpha.\tau) := \forall\alpha'.\pi[\alpha : \alpha'](\tau)$ for any α' which is fresh w.r.t. τ and $\pi(\tau)$.

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

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

$\pi(\lambda x_i^\sigma.t) := \lambda x_k^{\pi(\sigma)}.\pi[x_i^\sigma : x_k^{\pi(\sigma)}](t)$, for any variable $x_k^{\pi(\sigma)}$ of type $\pi(\sigma)$, which is fresh w.r.t. t and $\pi(t)$.

$\pi(\lambda\alpha.t) := \lambda\alpha'.\pi[\alpha : \alpha'](t)$ for any variable α' which is fresh w.r.t. t and $\pi(t)$. Recall that α is not free in the type of a free variable of t, hence $\pi[\alpha : \alpha'](t)$ makes sense.

We define now the notion of "*being good w.r.t. a substitution* π", which concerns the syntax of the bare abstraction F-types and F-terms (i.e. *not* up to α-equivalence). The interesting point is that, for each π, the expression of $\pi(t)$ can be simplified if t is good for π. Furthermore each abstraction type or term is α-equivalent to a type or term which is good for π.

Definition 63 *1.* $\forall\alpha.\tau$ *is* good for π *if* α *does not occur free in* $\pi(\beta)$ *for all* $\beta \neq \alpha$ *which are free in* τ.

2. $\lambda x_i^\sigma.t$ *is* good for π *if* $x_i^{\pi(\sigma)}$ *does not occur free in* $\pi(x_j^\tau)$ *for all* $x_j^\tau \neq x_i^\sigma$ *which are free in* t.

3. $\lambda\alpha.t$ *is* good for π *if* α *does not occur free in any* $\pi(\beta)$ *for all* $\beta \neq \alpha$ *which are free in* t, *nor in any* $\pi(x_j^\tau)$ *such that* x_j^τ *is free in* t.

Lemma 64 *Suppose that $\forall\alpha.\tau$ (resp. $\lambda x_i^\sigma.t$, resp. $\lambda\alpha.t$) is good for π. Then:*
1. $\pi(\forall\alpha.\tau) \simeq_\alpha \forall\alpha.\pi[\alpha:\alpha](\tau)$.
Furthermore $\pi[\alpha:\alpha_i](\tau) \simeq_\alpha \pi[\alpha:\alpha](\tau)[\alpha:\alpha_i]$ *for all* i.
2. $\pi(\lambda x_i^\sigma.t) \simeq_\alpha \lambda x_i^{\pi(\sigma)}.\pi[x_i^\sigma : x_i^{\pi(\sigma)}](t)$
Furthermore $\pi[x_i^\sigma : x_k^{\pi(\sigma)}](t) \simeq_\alpha (\pi[x_i^\sigma : x_i^{\pi(\sigma)}](t))[x_i^{\pi(\sigma)} : x_k^{\pi(\sigma)}]$ *for all* k.
3. $\pi(\lambda\alpha.t) \simeq_\alpha \lambda\alpha.\pi[\alpha:\alpha](t)$
Furthermore $\pi[\alpha:\alpha_i](t) \simeq_\alpha \pi[\alpha:\alpha](t)[\alpha:\alpha_i]$ *for all* i.

Lemma 65 *For all π, every bare type or term of the form $\forall\alpha.\tau$ or $\lambda x_i^\sigma.t$ or $\lambda\alpha.t$ is α-equivalent to a type or term which is good for π.*

Proof. For instance, $\forall\alpha.\tau \simeq_\alpha \forall\alpha'.\tau[\alpha := \alpha']$ for every α' not free in $\forall\alpha.\tau$. If we choose furthermore an α' which is not free in any $\pi(\beta)$ such that β is free in τ and distinct from α, we obtain that $\forall\alpha'.\tau[\alpha := \alpha']$ is good for π. ■

F Further proofs.

Proof of Lemma 40. We prove the second assertion by induction on σ (and for all compatible π, ρ at each step). The case where σ is a variable follows directly from the hypothesis.

- *Case 1.* Suppose $\sigma := \theta \to \tau$. Then:
  $type(\rho(\sigma)) = type(\rho(\theta \to \tau)) = type(\rho(\theta) \Rightarrow \rho(\tau))$
  $= arrow(type(\rho(\theta)), type(\rho(\tau)))$ (definition of *type* and nontriviality of $\rho(\tau)$).
  $\sim_{tp} arrow(\#(\pi(\theta)), \#(\pi(\tau)))$ (induction hypothesis plus commutativity of *arrow* with $\sim$)
  $= \#(\pi(\theta) \to \pi(\tau))$ (definition of *arrow*)
  $= \#(\pi(\theta \to \tau)) = \#\pi(\sigma)$ (inductive definition of π).
- *Case 2.* Suppose $\sigma := \forall\alpha.\tau$.
  By Lemma 65 in Appendix E we can assume that σ is good for π. Then:
  $type(\rho(\sigma)) = type(\rho(\forall\alpha.\tau)) = type(Q(Z \mapsto \rho[\alpha:Z](\tau)))$
  $= quant(\, i \in O \mapsto type(\rho[\alpha:C_i](\tau))\,)$ (definition of *type* and nontriviality of $\rho(\sigma)$ and of the argument of Q).
  $= quant(k)$ where k is the morphism defined by :
  $k(i) = type(\rho[\alpha:C_i](\tau))$. Now,
  $k(i) \sim_{tp} \#(\pi[\alpha:\alpha_i](\tau))$ (induction hypothesis plus Remark 39).
  Since $\forall\alpha.\tau$ is good for π, we have: $\#(\pi[\alpha:\alpha_i](\tau)) \sim_{tp} \#\pi[\alpha:\alpha](\tau)[\alpha:\alpha_i]$ (Lemma 64). Using now the definition of *quant* and that of $\pi(\forall\alpha.\tau)$ when $\forall\alpha.\tau$ is good for π (Appendix E) we have :
  $quant(k) \sim_{tp} \#(\forall\alpha.\pi[\alpha:\alpha](\tau)) = \#(\pi(\forall\alpha.\tau))$ (definition of *quant*).
  Thus $type(\rho(\forall\alpha.\tau)) \sim_{tp} \#(\pi(\forall\alpha.\tau))$ Q.E.D.

■

Proof of Lemma 42. The definition of $[-]_-$ is by recursion and by cases (Cond. I.4), using the left invertibility of $\langle -,-\rangle$ (Remark 14), and is quite straightforward. In the following we let $IF := IF_{Types}$ and we let $\phi(r;x)$ be another notation for $[x]_r$. We are looking for ϕ. To force the intended behavior of ϕ when x is the code of an F-type, it is enough that ϕ satisfies the following recursive equation:

$$\begin{aligned}\phi(r;x) = IF(&p_1(x), 0, r(p_2(x)),\\ &IF(p_1(x), 1, \phi(r; p_1(p_2(x))) \Rightarrow \phi(r; p_2(p_2(x))),\\ &\quad IF(p_1(x), 2, Q(X \mapsto \phi(j \mapsto IF(j, p_2(p_1(p_2(x))), X, r(j))\,; p_2(p_2(x))),\\ &\qquad \phi(r;x)\\ &\qquad\qquad))).\end{aligned}$$

The existence of such a morphism ϕ follows now from the existence of $Y_{O,(O\to Types)\to Types}$.

■

Proof of Lemma 54. The proof goes by induction on τ and shows at each step (as usual) that $\mathcal{R}^\tau_\Upsilon$ only depends of the value of Υ (and ρ, π) on the free variables of τ and has domain $\breve{O}_{\pi(\tau)}$. 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 (because of the compatibility of $Ap1$ and $Ap2$ with $\sim$), so we only have to prove the first two ones.

- *Case 1.* Suppose $\tau := \sigma \to \theta$.

  1. Suppose $x \in \breve{O}_{\pi(\sigma\to\theta)}$; we want to show $R^{\sigma\to\theta}_\Upsilon(x, f_{\rho(\sigma\to\theta)}(x))$.
     Suppose we have $\mathcal{R}^\sigma_\Upsilon(v,w)$ for some (v,w). Then, by the induction hypothesis, $v \in \breve{O}_{\pi(\sigma)}$ and $v \sim_{tr} v' := g_{\rho(\sigma)}(w)$. Also by the induction hypothesis, we have $\mathcal{R}^\theta_\Upsilon(Ap1(x,v'), f_{\rho(\theta)}(Ap1(x,v')))$. Since $Ap1$ commutes with $\sim_{tr}$ and because the third condition is true for $\mathcal{R}^\theta_\Upsilon$, still by the induction hypothesis, we have $\mathcal{R}^\theta_\Upsilon(Ap1(x,v), f_{\rho(\theta)}(Ap1(x,v')))$. By the definition of v' this can be rewritten as:
     $\mathcal{R}^\theta_\Upsilon(Ap1(x,v), [f_{\rho(\theta)} \circ Ap1(x) \circ g_{\rho(\sigma)}](w)\,)$, hence as:
     $\mathcal{R}^\theta_\Upsilon(Ap1(x,v), f_{\rho(\sigma)\Rightarrow\rho(\theta)}(x)(w))$. The conclusion now follows from $\rho(\sigma\to\theta) = \rho(\sigma) \Rightarrow \rho(\theta)$.
  2. Suppose $\mathcal{R}^{\sigma\to\theta}_\Upsilon(x,y)$; we want to prove :
     $x \in \breve{O}_{\pi(\sigma\to\theta)}$ and $x \sim_{tr} g_{\rho(\sigma\to\theta)}(y)$.
     For all $v \in \breve{O}_{\pi(\sigma)}$ we have :
     $\mathcal{R}^\sigma_\Upsilon(v, f_{\rho(\sigma)}(v))$ (by induction hypothesis on σ), hence :
     $\mathcal{R}^\theta_\Upsilon(Ap1(x,v), y(f_{\rho(\sigma)}(v)))$ (by definition of $\mathcal{R}^{\sigma\to\theta}_\Upsilon$).
     This forces $Ap1(x,v) \in \breve{O}_{\pi(\theta)}$. A first consequence is that $x \in \breve{O}_{\pi(\sigma\to\theta)}$. It is indeed enough to take for v the code of a well-formed

term of type $\pi(\sigma)$ (for example the code of a variable of type $\pi(\sigma)$) and to note that, since $Ap1(x,v)$ codes a well-typed term of type $\pi(\theta)$, necessarily, by invertibility of the pairing function, x codes a well-typed term of type $\pi(\sigma \to \theta) = \pi(\sigma) \to \pi(\theta)$.

The second consequence is that:

$g_{\rho(\theta)}(y(f_{\rho(\sigma)}(v))) \sim_{tr} Ap1(x,v)$ (by induction hypothesis on θ) $= Ap1(x)(v)$. Since this happens for all v, it follows (using Remark 33): $La1(\#(\pi(\sigma)), g_{\rho(\theta)} \circ y \circ f_{\rho(\sigma)}) \sim_{tr} La1(\#(\pi(\sigma)), Ap1(x))$. Now, $g_{\rho(\sigma\to\theta)}(y) = g_{\rho(\sigma)\Rightarrow\rho(\theta)}(y) := La1(type(\rho(\sigma)), g_{\rho(\theta)} \circ y \circ f_{\rho(\sigma)}) \sim_{tr} La1(\#(\pi(\sigma)), g_{\rho(\theta)} \circ y \circ f_{\rho(\sigma)})$ (by compatibility of ρ with π and commutativity of $La1$ with $\sim$ (Remark 32)). On the other hand : $La1(\#(\pi(\sigma)), Ap1(x)) \sim_{tr} x$ (Remark 34). Thus $g_{\rho(\sigma\to\theta)}(y) \sim_{tr} x$. Q.E.D.

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

  By Lemma 65 in Appendix E, we can assume that $\forall\alpha.\sigma$ is good for π.

  1. Let $x \in \breve{O}_{\pi(\forall\alpha.\sigma)}$. We want to show $\mathcal{R}^{\tau}_{\Upsilon}(x, f_{\rho(\forall\alpha.\sigma)}(x))$,
     i.e. $\mathcal{R}^{\tau}_{\Upsilon}(x, f_{\forall Z.\rho[\alpha:Z](\sigma)}(x))$.
     So, we have to show $\mathcal{R}^{\sigma}_{\Upsilon[\alpha:(X,T)]}(\, Ap2(x, type(X))\,,\, f_{\forall Z.\rho[\alpha:Z](\sigma)}(x)(X)\,)$ for all candidates (X,T). By definition of f_Y when $Y = Q(F)$ for some F, this amounts to show that:
     $\mathcal{R}^{\sigma}_{\Upsilon[\alpha:(X,T)]}(\, Ap2(x, type(X))\,,\, f_{\rho[\alpha:X](\sigma)}(Ap2(x, type(X))))$,
     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 $\mathcal{R}^{\forall\alpha.\sigma}_{\Upsilon}(x,y)$.

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

     $\mathcal{R}^{\sigma}_{\Upsilon[\alpha:T]}(Ap2(x, type(X)), y(X))$ for all candidates (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 52), 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, $\forall\alpha.\sigma' = \forall\alpha.\pi[\alpha : \alpha](\sigma) = \pi(\forall\alpha.\sigma) = \pi(\tau)$ (Lemma 64) hence x is the code of a well-typed term of type $\pi(\tau)$, i.e. $x \in \breve{O}_{\pi(\tau)}$, 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 34). Q.E.D.

■

Lemma 66 *For all compatible Υ, ρ, π, for all α, σ, τ we have :* $\mathcal{R}^{\tau[\alpha:\sigma]}_{\Upsilon} = \mathcal{R}^{\tau}_{\Upsilon[\alpha:(X,T)]}$ *with* $(X, T) := \Upsilon(\sigma)$.

Proof. Straightforward induction on τ. ■

Proof of Proposition 57. 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 (Definition 45), 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 $t := \lambda x^{\sigma}_i.u : \sigma \to \tau$, with $u : \tau$. By Lemma 65 in Appendix E we can assume that $\lambda x^{\sigma}_i.u$ is good for π (since we are only interested in the α-equivalence class of $\pi(t)$).

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

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

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

  $Ap1(\#(\pi(\lambda x^{\sigma}_i.u)), \#t') = \#((\pi(\lambda x^{\sigma}_i.u))t')$ (definition of $Ap1$) $= \#(\pi'(u))$, where $\pi' := \pi[x^{\sigma}_i : t']$ and π' acts like π on types (using Lemma 64). Similarly:

  $(\rho(\lambda x^{\sigma}_i.u))(w)) = \rho(u[x^{\sigma}_i : w]) = \rho'(u)$, where $\rho' := \rho[x^{\sigma}_i : w]$ 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.

  Once more we assume that $\lambda\alpha.u$ is good for π.

  The aim is to prove $\mathcal{R}^{\forall\alpha.\tau}_{\Upsilon}$ ($\#(\pi(\lambda\alpha.u))$, $\rho(\lambda\alpha.u))$, that is to say: to prove:

$\mathcal{R}^{\tau}_{\Upsilon[\alpha:(X,T)]}\,(\,Ap2(\#(\pi(\lambda\alpha.u)), type(X)),\,(\rho(\lambda\alpha.u))(X)\,)$ for all candidates (X,T).

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) by Remark 52. For defining ρ', π' at the level of term variables we just change π and ρ on those x_i^{θ} such that α is free in θ. We set $\pi'(x_i^{\theta}) = x_i^{\pi'(\theta)}$ and $\rho'(x_i^{\theta}) = f_{\rho'(\theta)}(x_i^{\pi'(\theta)})$, which obviously preserve the strong compatibility condition. Now,

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

$\sim_{tr} \#(\pi[\alpha : \sigma]((\lambda\alpha.u)\alpha)) = \#(\pi'((\lambda\alpha.u)\alpha)) \sim_{tr} \#\pi'(u)$. 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 $\mathcal{R}^{\tau}_{\Upsilon'}(\,\#(\pi'(u)),\,\rho'(u))\,$, which follows from the 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 :

  $\mathcal{R}^{\tau[\alpha:\sigma]}_{\Upsilon}(\,\#(\pi(u\sigma)),\,\rho(u\sigma))$. 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 $type(X) \sim_{tp} \#(\pi(\sigma))$ by compatibility of π, ρ. Furthermore $\mathcal{R}^{\tau[\alpha:\sigma]}_{\Upsilon} = \mathcal{R}^{\tau}_{\Upsilon_{[\alpha:T]}}$ where $T = \mathcal{R}^{\sigma}_{\Upsilon}$ (Lemma 66).

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

■

References

[1] M. Abadi, G. D. Plotkin, A logic for parametric polymorphism, in TLCA'93, SLNCS n^0 664, 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, June 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 full continuous model of polymorphism", technical report, University of Turin, 1997, available on the authors' web pages.

[6] S. Berardi, Retractions on dI-Domains as a Model for Type:Type, Information and Computation 94, p.204-231, 1991.

[7] S. Berardi, C. Berline, Building continuous webbed models for System F, preprint, Octobre 1998 revised September 2000, available on the authors' web pages.

[8] S. Berardi, C. Berline, Easy models for $F_\omega Y$, Calculus of Construction, and $kinds = types$, in preparation.

[9] C. Berline, Rétractions et Interprétation 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. Flagg, κ-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, Interprétation fonctionnelle et élimination des coupures de l'arithmétique d'ordre supérieur, Thèse de doctorat d'état, Université Paris 7, 1972.

[18] J.Y. Girard, The system F of variable types, fifteen years later, Theoretical computer Science, vol.45, p.159-192, 1986.

[19] G. Longo, Parametric and Type-dependent Polymorphism, Fundamenta Informaticae, vol. 22, 1-2, p.69-92, 1995.

[20] 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.

[21] G. Longo and E. Moggi, Constructive natural deduction and its ω-set interpretation, Math. Struct. in Comput. Science, vol. 1, pp. 215-254, 1991.

[22] N. J. McCracken, An Investigation of a Programming Language with a Polymorphic Type Structure, PhD Dissertation, Syracuse University, New York, 1979.

[23] N. J. McCracken, A Finitary Retract Model for the Polymorphic λ-calculus, unpublished, 1984.

[24] P. O'Hearn, Parametric Polymorphism, in: M. P. Fiore, A. Jung, E. Moggi, P. O'Hearn, J. Riecke, G. Rosolini and I. Stark, Domains and Denotational Semantics : History, Accomplishments and Open Problems, Bulletin of EATCS, vol.59, p.227-256, 1996.

[25] A.M. Pitts, Polymorphism is set-theoretic, constructively. In: Category Theory and Computer Science, Lecture Notes in Computer Science 283, Springer, 1987.

[26] G.D. Plotkin, Lambda-definability and Logical Relations, Technical report SAIRM-4, School of artificial intelligence, University of Edinburgh, 1973.

[27] G. D. Plotkin, Lambda-definability and the full-type hierarchy, in: J.P. Seldin and J.R. Hindley, eds., To H.B. Curry: Essays on Combinatory Logic, Lambda-calculus and Formalism, Academic Press, New-York, 1980.

[28] G.D. Plotkin, A power domain for countable nondeterminism, in: Lecture Notes in Computer Science vol. 140, ICALP'82, p.418-428, Springer Verlag, Berlin, 1982.

[29] J.C. Reynolds, Towards a Theory of Type Structure. In Colloque sur la Programmation, LNCS 19, p.408-425, Springer-Verlag, 1974.

[30] 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.

[31] 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.

[32] D. Scott, Data Types as Lattices, S.I.A.M. J. of Computing 5, p. 522-587, 1976.

[33] D. Scott, A Space of retracts, manuscript, 1980.

[34] R. Statman, Completeness, Invariance and λ-definability, J. of Symb. Logic, vol.47, n°1, 1982.

[35] 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
URL: www.di.unito.it/~stefano

C. Berline, CNRS-Université Paris 7
UFR de Mathématiques (case 7012)
Université Paris 7
2 place Jussieu,
75251 Paris cedex 05, FRANCE.
e-mail: berline@pps.jussieu.fr
URL: www.logique.jussieu.fr/www.berline