Realizability structures play a major role in the metamathematics of intuitionistic systems and they are a basic tool in the extraction of the computational content of constructive proofs. Besides their rich categorical structure and effectiveness properties provide a privileged mathematical setting for the semantics of data types of programming languages. In this paper we emphasize the modelling of recursive definitions of programs and types. A realizability model for a language including Girard’s system F and an operator of recursion on types is given and some of its local properties are studied. © 1991 Academic Press, Inc.

INTRODUCTION

A few historical remarks are appropriate to understand the relevance of realizability structures in logic and to discuss the possibility of building on them mathematical theories of computation well-suited for the study of programming languages. In his pioneering work Kleene (1945) proposed an interpretation of intuitionistic arithmetic over the structure $(\omega, \cdot)$ of natural numbers equipped with a partial operation of application defined as: $n \cdot m = [n](m)$, where $[ ]$ is a Godel-numbering of the partial recursive functions. Such interpretation provides a standard link between constructive logic and classical recursion theory. The relevant property of $(\omega, \cdot)$ is that of being a partial combinatory algebra (see, e.g., Barendregt, 1984) hence providing the basic operations of functional computing, i.e., abstraction, application, pairing, projection etcetera.

A variety of systems have been interpreted over these algebraic structures (Troelstra, 1973; McCarty, 1985) with a wide fallout of metamathematical results, in particular the consistency of intuitionistic theories with a number of principles, e.g., Church’s thesis, Markov’s principle, uniformity principle, axiom of dependent choice (see Troelstra and Van Dalen, 1988, for an up-to-date account). A first important application (not considered here) of
realizability methods to computer science arises in the area of program synthesis. Various forms of realizability (see Hayashi and Nakano, 1988, for a discussion of the pragmatic) represent a well-suited tool for extracting the computational content of constructive proofs.

The basic ideas of realizability also reappear in various approaches to an abstract theory of computation, notably and foremost in Malcev’s category of numbered sets. An enumerated set is a set $A$ equipped with an onto mapping $e_A : \omega \rightarrow A$ (i.e., an enumeration), and a morphism in the category is a set theoretic function $f : A \rightarrow B$ s.t. $\exists \phi \in \omega \forall n \in \omega f(e_A(n)) = e_B(\phi \cdot n)$. The search for categories with stronger closure properties led to the inventions of the category of partial equivalence relations (appearing in various forms from Kreisel, 1959, to Scott, 1976) and the effective topos (Hyland, 1982).

Now it is a well-known result (Moggi, 1986; Hyland, 1988; Carboni et al., 1987; Longo and Moggi, 1988) that the category of partial equivalence relations (per) over a partial combinatory algebra (pea) can be seen as an internal small “complete” category of the effective topos and indeed of other simpler structures.

This property leads to interpretations of (impredicative) higher order lambda calculi that imply their logical and equational consistency.

The richness of the categorical structure and the computational intuition underlying the notion of morphism suggest the adoption of a realizability universe as a setting for the semantics of data types of programming languages. Nevertheless an aspect of programming languages, that is not adequately taken into account in the mentioned categories, is that of divergent computations. Indeed the introduction of unbounded iterations is a canonical choice in the design of a programming language as the issues of flexibility and expressivity of the formalism usually have priority on the complication of the programming logic.

In this paper we consider the problem of interpreting fixed point combinators for terms and types over a realizability model of higher order lambda calculi; in particular we are interested in extensions of the second-order lambda calculus (Girard, 1972; Reynolds, 1974, Sect. 2).

Dependent types are not considered for various rather pragmatical reasons, namely: (i) their categorical treatment is still under development, (ii) their introduction would entail difficulties that are somehow orthogonal to the problems considered here, (iii) typed languages with divergent computations and dependent types are hard to type-check.

Indeed we are interested in $\lambda$-calculi that admit a static type checking discipline in which type information disappears at runtime. Realizability models are particularly well-suited for such languages as the two stages of compilation and execution are clearly separated in the interpretation.

In particular we will consider per’s over a $\lambda$-model (Section 1) rather than over a pca; in this case it is easy to relate the interpretation of a typed
term with that of its type-free "erasure." Indeed the interpretations of the typed terms are equivalence classes of type-free denotations; intuitively such equivalence classes are composed of exactly those elements that under the type restrictions behave in the same way. This observation will be a starting point for the study of the local structure of the models (Section 6). In particular we give sufficient conditions for (i) equating typed terms, (ii) determining "unsolvable types," and (iii) analysing type containments.

The second basic and non-standard assumption on the realizability structure concern the satisfaction of certain topological properties (working in subcategories of complete partial orders). This is hardly an original approach as it was introduced by Dana Scott some twenty years ago, the challenging point is rather to find an harmonic combination of the topological structures with the categorical constructions of the per-models (Sections 3–5).

Although the paper is essentially about mathematical structures the inspiring ideas come from the need for an explanation and elucidation of certain linguistic constructs in functional programming. Besides the study of the local and global properties of the models provides non-trivial principles to reason about and to optimize programs. In view of the growing interest for the notion of inheritance we should also mention that the formal theory underlying the analysis of type-containments (6.3; see also Amadio, 1989) extends to recursive types the one proposed in Bruce and Longo, 1988).

1. SOME STRUCTURES FOR COMPUTATIONS

In this paragraph the categories more relevant for our goals are introduced and an analysis of their properties and relationships is undertaken. Familiarity with elementary category theory and the construction of the $D_\infty \lambda$-model is assumed.

1.1. Realizability Structures

When working in the category of complete partial orders (cpo) $C^C$ denotes the cpo of continuous functions over the cpo $C$.

Let $D$ be a cpo s.t. $D^D$ is a retract of $D$ by means of the pair $(i: D^D \to D, j: D \to D^D)$ satisfying: $j \circ i = id_{D^D}(\ast)$. Such a reflexive object is a $\lambda$-model (see, e.g., Barendregt, 1984). The natural operation of application on $D$ is defined as $f \cdot d = j(f)(d) (=_{ab} f d)$.

From the previous assumptions it is possible to prove that $D \times D$ is a retract of $D$ by means of the pair $[\ ]: D \times D \to D$, $p: D \to D \times D$ satisfying $p \circ [\ ] = id_{D \times D}$. 
As usual \([x, y] \equiv_{abf} \lceil((x, y))\) and \(p_i x \equiv_{abf} \pi_i(p(x))\) (\(i = 1, 2\); \(\pi_i\) cartesian projection).

Some of the constructions will take place over a \(\lambda\)-model built by means of an inverse limit construction (Scott, 1972), as usual we denote such \(\lambda\)-model with \(D_{\infty}\).

Consider a functor \(F\) on the category of cpo's and projection pairs that on objects is defined as \(F(D) = \lambda (D^D) + (D \times D) + \text{At}\), where: \(+\) is the coalesced sum on cpo's and \(\text{At}\) is a non-trivial cpo of atomic values. On morphisms (i.e., projection pairs \((i, j): D \to E \equiv_{abf} i: D \to E, j: E \to D, j \circ i = id \land i \circ j \leq id\) \(F\) is defined by means of an obvious combination of the standard functors \(\lambda D \cdot D^D, \lambda D \cdot D \times D, \) and \(\lambda D \cdot \text{At}\).

\(D_{\infty} \equiv (D^D_{\infty}) + (D_{\infty} \times D_{\infty}) + \text{At}\) is the initial fixed-point of the functor \(F\), i.e., the initial \(F\)-algebra. More concretely, it is built out as the colimit of the \(\omega\)-diagram:

\[
D_0 = \{\bot\}, \quad D_n + 1 = (D_n^D) + (D_n \times D_n) + \text{At} \quad \text{and the uniquely determined projection pairs} \quad (i_{n,n+1}, j_{n,n+1}): D_n \to D_{n+1}.
\]

Conventions and basic properties. Let \((i_n, j_n)\) denote the projective pair between \(D_n\) and \(D_{\infty}\). If \(d \in D_{\infty}\) then \(d_n = i_n(j_n(d))\). We refer to the set \(\{d \in D_{\infty} | \exists n \geq 0 \ d = d_n\}\) as the collection of approximating elements. In particular, given an element \(d \in D_{\infty}\) its approximating elements are \(\{d_n/n \geq 0\}\).

The name seems appropriate as \(d = \bigcup_{n \geq 0} d_n\).

Of course, this \(\lambda\)-model satisfies the previous condition (*); in particular each element \(d \in D_{\infty}\) can be seen, up to isomorphism and exclusively, as either a function or a pair or an atomic value with the exception of \(\bot\).

Let \(k: D_{\infty} \to (D^D_{\infty}) + (D_{\infty} \times D_{\infty}) + \text{At}\) be such an isomorphism then (in a semi-formal notation):

\[
j(d) = \lambda \quad \text{if} \quad "k(d) is a function" \quad \text{then} \quad k(d) \land \lambda \cdot \bot,
\]

\[
p(x) = \lambda \quad \text{if} \quad "k(x) is a pair" \quad \text{then} \quad k(x) \land (\bot, \bot).
\]

The related injections \(i: D^D_{\infty} \to D_{\infty}\) and \([, ]: D_{\infty} \times D_{\infty} \to D_{\infty}\) are obvious.

Also recall that the application defined as \(f \cdot e = \lambda j(f)(e)\) satisfies \(f \cdot e = \bigcup_{n \in \omega} f (f_{n+1})(e_n)\).

Keeping in mind the morphism component of the functor \(F\), it is not difficult to prove that

\[
d_{n+1} e = d_{n+1} e_n = (de_n)_n \quad \text{and} \quad [x, y]_{n+1} = [x_n, y_n].
\]

1.2. Objects

We define the objects of certain related categories, they will be the denotations of the types of our languages.
1.2.1. **Definition.** Let $A$ denote a binary relation over $D$. $A$ is

1. a *partial equivalence relation* (per) if it is symmetric and transitive;
2. *pointed* if $(\bot, \bot) \in A$;
3. *complete* if given $X$ directed in $A \cup X \in A$;
4. *uniform* if $dAe \Rightarrow \forall n < \omega dAe_n$,

where it is intended that the realizability structure is a $\lambda$-model in cpo for cases (2), (3) and the previously defined $D_\infty$ for case (4).

**Terminology.** If we assume (1) and combine the three remaining conditions we get eight kinds of per's. Being interested in fixed-points at all types we will *focus on pointed categories*. In particular we will consider $cper \equiv_{abf}$ pointed, complete per, and $cuper \equiv_{abf}$ pointed, complete, uniform per. For our analysis we will also refer to $per \equiv_{abf}$ pointed per and $uper \equiv_{abf}$ pointed, uniform per. Perhaps $cpper$ etcetera would be more appropriate acronyms, however, since the non-pointed cases are not considered here, we prefer to follow the tradition coming from domain theory where we speak of cpo's and not of cppo's. By no means does this imply that the non-pointed cases should be neglected. Sometimes we subscript the acronym with the realizability structure, e.g., $per_D$.

**Remark.** Note that cper's anduper's are never empty. Besides if $A$ is a cper then $A$ and its domain (i.e., $\{d/(d, d) \in A\}$) are cpo's respectively in $D \times D$ and $D$. Also observe that each equivalence class is closed under sup's of directed sets.

1.3. **Morphisms**

Let $dAe \equiv_{abf} (d, e) \in A$; $[d]_A$ denotes the equivalence class of $d$ in $A$. $D/A = \{[d]_A/dAd\}$ (the quotient space). Denote with $C$ one of the eight categories that can be obtained by considering as objects per's satisfying any combination (possibly empty) of the properties of pointedness, completeness and uniformity and the following as *morphisms*: for $A, B$ in $C$

$$C[A, B] = \{f: D/A \rightarrow D/B/\exists \phi \in D \forall d \in D dAd \rightarrow \phi d \in f([d]_A)\}.$$  

The morphisms of the category $C$ are the *set theoretic functions* between the quotient spaces that are *realized* by some representable function.

1.4. **Closure Properties**

1. $C$ is a cartesian closed category.
2. The objects of $C$ are closed under arbitrary set-theoretic intersections.
Proof. The proof proceeds by showing that pointedness, completeness, and uniformity are independent properties of a per.

(1) A terminal object is \( \{ (\bot, \bot) \} \). Define the product as: \( dA \times Be \) iff \( p_1.dAp \) and \( p_2 dBp \). This is clearly a per. Suppose \( \bot A \bot \) and \( \bot B \bot \) then \( \bot A \times B \bot \), since \( p \bot = (\bot, \bot) \). Suppose \( A, B \) are complete; let \( X \) be a directed set, \( X \in A \times B \).

Define \( X_i = \{ (p_i.x, p_i.y)/(x, y) \in X \} \) for \( i = 1, 2 \). Then \( X_1 \) and \( X_2 \) are directed sets respectively in \( A \) and \( B \). Hence \( (d_1, e_1) = \bigcup X_i \in A \) and \( (d_2, e_2) = \bigcup X_i \in B \). Now just observe that the function \( P_i = \lambda (x, y) \cdot (p_i.x, p_i.y) \) is continuous hence: \( P_i(\bigcup X) = \bigcup P_i(X) = \bigcup X_i = (d_i, e_i) \), for \( i = 1, 2 \). Suppose \( A \) and \( B \) are uniform. Some case analysis is needed; in particular if \( x = [x_1, x_2] \) and \( y = [y_1, y_2] \), use the fact: \( [x, y]_{n+1} = [x_n, y_n] \).

As an exercise define the projections and show that \( A \times B \) is a categorical product. Define the exponent as: \( fBAg \) iff \( \forall de, dAe \Rightarrow fdBge \). \( B^A \) is trivially a per. Note that \( lBAl \) implies \( lBAl \) as \( f(\bot) = \lambda x. \bot \). Suppose \( A, B \) complete and let \( X = \{ (f_i, g_i)/i \in I \} \) be a directed set in \( B^A \). Note that, e.g., if \( dAd \) then \( (\bigcup_{i \in I} f_i) d = \bigcup_{i \in I} (f_i.d) \in \text{Dom}(B) \). Besides \( \forall de, dAe \Rightarrow \{ (f_i.d, g_i.e)/i \in I \} \) is a directed set in \( B \). Now just observe that: \( \bigcup X = (\bigcup_{i \in I} f_i, \bigcup_{i \in I} g_i) \in B^A \Rightarrow \forall de(dAe \Rightarrow (\bigcup_{i \in I} (f_i.d), \bigcup_{i \in I} (g_i.e)) \in B) \).

Assume \( A, B \) uniform and \( fB^Ag, f_{n+1}B^Ag_{n+1} \Rightarrow \) \( \forall de(dAe \Rightarrow f_{n+1}dBg_{n+1}e) \).

Now \( dAe \Rightarrow d_n.Ae_n \Rightarrow fd_n.Bge_n \Rightarrow (fd_n)_{n+1}B(ge_n)_{n+1}f_{n+1}dBg_{n+1}e \), since \( A \) and \( B \) are uniform and \( f_{n+1}d = (fd_n)_{n+1} \). The case \( n = 0 \) is trivial.

Complete the proof by constructing a natural isomorphism between the hom-functors \( C[_, A, B] \) and \( C[_, B^A] \).

(2) The closure under intersection is immediate. Q.E.D.

Notes. Let us concentrate for a moment on cuper's. There are no mathematical problems in adding other standard data-type constructors like coalesced sum and lifting modulo a suitable definition of the underlying \( D_x \) \( \lambda \)-model.

It is also possible to give slightly more refined interpretations of the data types, e.g., following the so-called \( F \)-semantics define the exponent of \( A, B \) as \( B^A \cap F^2 \), where \( F = \{ d/d = \lambda j(d) \} \) (observe that \( F \) is a cuper and \( B^A \cap F^2 \cong B^A \)).

1.5. The Frame Category

The category \( C \) lives inside a simple set-theoretic realizability universe (Carboni et al., 1987; Longo and Moggi, 1988) of \( D \)-sets. This category is a natural frame to speak about the completeness properties of the category \( C \) (Hyland, 1988) and to interpret higher order extensions of the second order \( \lambda \)-calculus (e.g., Girard's system \( F_0 \)). It is also an essential tool to show that our interpretation satisfies certain categorical properties, e.g., the
interpretation of the second-order quantification is, following Lawvere, the right adjoint of the diagonal functor (see 1.6). Nevertheless the reader can safely ignore all these important facts as they are not used in the following sections.

1.5.1. **Definition.** The large category $\textbf{D-set}$ has objects: $Ob_{\textbf{D-set}} = \{(S, \vdash s) \mid \vdash s \subseteq D \times S, S \text{ set and, } \forall s \in S, \exists d \in Dd \vdash s\}$ and as morphisms:

$$\textbf{D-set}[(S, \vdash s), (T, \vdash t)] = \{F: S \to T/\exists \phi \in D, \forall s \in S, \forall d \in D, d \vdash S \Rightarrow \phi d \vdash F(s)\}.$$

1.5.2. **Facts** (for category theorists). $\textbf{D-set}$ is a (locally-) cartesian closed category, has all finite limits, and is a quasi-topos. It can be seen as the full subcategory of the $(\sim \sim \sim)$ separated objects of the effective topos built over $D$. Also the category of sets can be fully ad faithfully embedded in $\textbf{D-set}$ by a means of a functor $A$ that to a set $S$ associates the $\textbf{D-set} (S, D \times S)$.

$C$ is equivalent to the full sub-ccc $\textbf{M}_C$ ($M$ for modest, after Scott) of the objects $(S, \vdash s)$ satisfying:

(i) $d \vdash s$ and $d \vdash t \Rightarrow s = t$ (i.e., $\vdash s$ is a single-valued relation from $D$ into $S$),

(ii) the per $A$ defined as: $dAe \iff \exists s, d \vdash s$ is an object of $C$.

Besides $C$ can be seen as an internal category of $\textbf{D-set}$.

A quite relevant issue, raised by one of the referees, concerns the internal definability of the full subcategories of per's considered. This is a fascinating question open to further investigation. Note that in $\textbf{c(u)}$ per we lose the existence of equalizers and initial object; this is an unavoidable compromise towards an interpretation of a fixed point combinator. Indeed a ccc with fix-points and an initial object is inconsistent (Huwig and Poigne, 1986).

1.6. **Models for Higher Order Lambda Calculi**

Various definitions of models for the pure second-order lambda calculus have appeared in the literature. Bruce and Meyer (1984) propose an elementary definition based on the notion of “second-order frame.” The approach is not very informative w.r.t. the “structural properties” needed to construct a second-order model. Moggi (1986) and Seely (1988) give categorical definitions based, respectively, on notions of internal and indexed category theory. Every model in the sense of Moggi can be translated in a model according to Seely (see, e.g., Asperti and Longo 1990). We only give the frame for the Bruce–Meyer interpretation; however, we stress that the structures considered lead to models also in the natural categorical sense.
1.6.1. The algebra of polymorphic types. Define:

The collection of types: \( T = \Delta \text{Ob}_C \)

The function space constructor: \( A \Rightarrow B = \Delta B^A \)

The operators over types: \( [T \rightarrow T] = \Delta \{ F/F: T \rightarrow T \text{ in Set} \} \)

The indexed product: \( \prod F = \Delta \cap_{A \in T} F(A) \).

1.6.2. Remark. It has been pointed out by several people that the interpretation of an indexed product as an intersection is a consequence of a generalized version of the uniformly principle:

\((\text{UP}) \forall X \exists n, R(X, n) \rightarrow \exists n \forall X, R(X, n)\), where \( n \) ranges over \( N \) (natural numbers) and \( X \) over \( \wp(N) \). To have some feeling of this fact consider our interpretation as living inside the category \( D\text{-set} \).

Then \( T \), the collection of types, is represented by the \( D\text{-set} \( \Delta(\text{Ob}_C) \) and types are modest sets (see 1.5.2). Assume that \( F \), the type constructor, is a constant function, say \( \forall A F(A) = B, B \) non-trivial, then we should have (with some abuse of notation) \( \prod F = B \cong B^T \). This is in general false in Set but holds in \( D\text{-Set} \) as \( B^T \) is inhabited only by constant functions.

1.6.3. The second-order extensional model.

Domains: \( \forall A \in T, \text{Dom}_A = \Delta D/A \)

Term application: \( \forall A, B \in T, \cdot_{A,B}: \text{Dom}_A \Delta B \times \text{Dom}_A \rightarrow \text{Dom}_B \)

\( [f]_{B^A} \cdot_{A,B} [d]_A = \Delta [fd]_B \)

Type application: \( \forall F \in [T \rightarrow T], \cdot_F: \text{Dom}_{\prod(F)} \times T \rightarrow \bigcup_{A \in T} \text{Dom}_{F(A)} \)

\( [f]_{\prod(F)} \cdot_F A = \Delta [f]_{F(A)} \).

Note that the operations of application are extensional; hence the interpretation of an abstraction is uniquely determined by its applicative behavior. This is the key observation to show that this is a second-order (extensional) \( \lambda \)-model (more details in, e.g., Breazu-Tannen and Coquand, 1988).

2. A Fixed-Point Extension of the Second-Order \( \lambda \)-Calculus and Its Interpretation

The syntax of a second-order lambda calculus extended with a fixed-point combinator on terms is defined, it represents the core of a functional
language with explicit polymorphism. The interpretation of the language in
the category of c(u)per's is specified; some local properties of the model
are considered in Section 6. The operational properties of the calculus are
studied in Amadio (1988) w.r.t. head reduction mechanism of evaluation.

2.1. Types

Denote with $T_{var}$ a countable collection of types variables $\{t, s, \ldots\}$
then the collection $T_p$ of types is inductively defined as follows:

$$ T_p ::= T_{var} \mid (T_p \to T_p) \mid (\forall T_{var} \cdot T_p) $$

2.2. Well-Typed Terms

Assume for each type $\alpha$ a countable collection of variables $x^a, y^a, \ldots$. Of
course $x^a$: $\alpha$ (read as $x^a$ has type $\alpha$). The system is presented in a natural
deduction style:

$$
\begin{align*}
(\to I) & M: \beta \Rightarrow (\lambda x^a \cdot M): \alpha \to \beta \\
(\to E) & M: \alpha \to \beta, N: \alpha \Rightarrow (MN): \beta \\
(\forall I) & M: \alpha \Rightarrow (\forall t \cdot M): \forall t \cdot \alpha^1 \\
(\forall E) & M: \forall t \cdot \alpha \Rightarrow (M\beta): [\beta/t] \alpha \\
(Fix) & M: \sigma \Rightarrow \sigma \Rightarrow (Y_\sigma M): \sigma,
\end{align*}
$$

Assume the usual $\beta(\eta)$ rules of conversion and the fixed-point rule:

$$ YM = M(YM). $$

2.3. A Fixed-Point Operator

A least fixed-point operator Fix can be defined in the $\lambda$-model $D$ as
follows: $\text{Fix} = \bigcup_{n<\omega} [\lambda z \cdot z^n \perp]^D$, where: $[\cdot]^D$ is the interpretation of
closed, type free $\lambda$-terms in the $\lambda$-model $D$ and $[\lambda]^D = \downarrow_{\rho}$. The reader can check that:

(i) Fix is well defined,

(ii) Fix $f = f(Fix f) = \bigcup_{n<\omega} f^n \perp$ is the least fixed point of $f$.

Working in cper we can also define a fixed-point operator on each
semantic type; for each $A$ cper let:

$$ \text{Fix}_A = [\text{Fix}]_{(A \Rightarrow A) \Rightarrow A}. $$

We hasten to state that this is a correct definition.

2.3.1. Proposition. Let $A$ be a cper, then:

(1) Fix$_A$ is well defined, i.e., $[\text{Fix}]_{(A \Rightarrow A) \Rightarrow A}$ is not empty.

(2) Given $F \in \text{cper}[A, A]$, Fix$_A F = F(\text{Fix}_A F)$.

$^1 t$ not free in the type of a variable free in $M$. 

643/91/1-5
Proof. (1) Observe that: \( \{ ([\lambda z \cdot z^n]_n, [\lambda z \cdot z^n]_n) \}_{n < \omega} \) is a chain in \((A \Rightarrow A) \Rightarrow A\) as: \( \forall n \forall f \forall g \cdot A^4 \cdot A \Rightarrow f^n \cdot A \cdot g^n \cdot A \). Then by completeness: \( \text{Fix} \in \text{Dom}( (A \Rightarrow A) \Rightarrow A ). \)

(2) We just recall that: \([f]_B \cdot A, B \cdot [d]_A = d \cdot [fd]_B\), this allows to exploit the properties of Fix. Q.E.D.

2.4. Interpretation

The following interpretation works for all the categories considered with the exception of the Y clause where \( C \) can be only cper or cuper. Given a type-environment: \( \eta : T_{\text{var}} \rightarrow T \), the type interpretation is defined as follows:

\[
[t]_\eta = \eta(t)
\]

\[
[\alpha \rightarrow \beta]_\eta = [\alpha]_\eta \Rightarrow [\beta]_\eta \quad (= [\beta]^{[\alpha]}_\eta)
\]

\[
[\forall t \cdot \alpha]_\eta = \prod_{A \in T} \cdot [\alpha]_{\eta[A/t]} \quad (= \cap_{A \in \text{objc}} [\alpha]_{\eta[A/t]}).
\]

Let \( \eta : T_{\text{var}} \rightarrow T \), \( \rho : \text{TypedVar} \rightarrow \bigcup_{A \in T} \text{Dom}_A \), where \( \rho(x^\alpha) \in \text{Dom}_{[\alpha]_\eta} \). Then w.r.t. the well-typed term the term interpretation is inductively defined as follows:

\[
[x^\alpha]_{\rho\eta} = \rho(x^\alpha)
\]

\[
[\lambda x^\alpha \cdot N^\beta]_{\rho\eta} = \{ \phi \in D / (d[\alpha]_\eta \Rightarrow \phi d \in [N^\beta]_{\rho([(d[\alpha]_\eta) / x^\alpha]_\eta)}) \}
\]

\[
[N^\alpha \rightarrow \beta \cdot P^\alpha]_{\rho\eta} = [N^\alpha \rightarrow \beta]_{\rho\eta} \cdot A, B \cdot [P^\alpha]_{\rho\eta},
\]

where \( A = [\alpha]_\eta \) and \( B = [\beta]_\eta \)

\[
[\lambda t \cdot N^\beta]_{\rho\eta} = \{ \phi \in D / \forall A \in T, \forall d \in D, \phi d \in [N^\beta]_{\rho\eta[A/t]} \}
\]

\[
[M^\forall t \cdot z^\beta]_{\rho\eta} = [M^\forall t \cdot z]_{\rho\eta} \cdot F \cdot [\beta]_\eta,
\]

where \( F = \lambda A \in T \cdot [\alpha]_{\eta[A/t]} \).

\[
[Y_\alpha M^\alpha \rightarrow z]_{\rho\eta} = \text{Fix}_A \cdot B, A \cdot [M^\alpha \rightarrow z]_{\rho\eta},
\]

where \( A = [\alpha]_\eta \) and \( B = [(\alpha \rightarrow x) \rightarrow x]_\eta \).

Note. Both at syntactic and semantic levels one can introduce a notion of environment; working in a concrete category we prefer this more manageable notation.

3. COMPLETIONS AND INDUCTIVE TYPES

In this paragraph we concentrate on an analysis of the relationships among the subcategories of \( \text{per}'s \) introduced in Section 1. In particular, we
show that $c_{per_D}$ is reflective in $per_{\bot_D}$ and $uper_{D\alpha}$ is reflective in $uper_{D\alpha}$ (3.3). A first application of this result follows concerning an interpretation of inductive types.

Let us recall that a subcategory $C$ of $C'$ is called reflective in $C'$ (see, e.g., MacLane, 1972) when the inclusion functor $\text{Incl}: C \to C'$ has a left adjoint. Equivalently one has to build a functor $F: C' \to C$ and a natural isomorphism between the hom-functors $C[F-, -]$ and $C'[-, \text{Incl}-]$.

Suppose $C \subset C' \subset per$, where $\subset$ stands for subcategory and $C, C'$ are two of the subcategories of $per$ introduced in Section 1. In trying to define $F: C' \to C$ we can be tempted to set: $F(A) = \bigcap \{B/B \in Ob_C \text{ and } B \supseteq A\}$. This is well defined as $Ob_C$ is closed under arbitrary set-theoretic intersections and $D^2 \in Ob_C$. However, it is not always the case that $F$ can be extended to a functor left adjoint of the inclusion functor, e.g., consider the case $C' \equiv c_{per}$ and $C \equiv c_{uper}$.

In the following we consider two favourable situations for $C'$ being $per_{\bot}$ or $uper$ and $C$ being, respectively, $c_{per}$ or $c_{uper}$. For the sake of brevity set for $A \in per_{\bot} A = \bigcap \{B/B \in c_{per}, B \supseteq A\}$.

In the first place we need an inductive characterization of $A$. Define for $R$ binary relation on $D$:

$\text{Sup}(R) = \bigcap \{X | X \text{ directed in } R\}$ (Directed closure).

$\text{TC}(R) = \bigcap \{S | S \supseteq R \text{ and } (xSy \land ySz \Rightarrow xSz)\}$ (Transitive closure).

Observe that:

(a) Both $\text{Sup}$ and $\text{TC}$ preserve symmetry and pointedness.

(b) $\text{TC} \circ \text{Sup}: per_{\bot} \to per_{\bot}$. Whereas in general $\text{Sup}(A)$ is neither transitive nor complete.

Given $A \in per_{\bot}$ let $A' = \text{TC}(\text{Sup}(A))$.

3.1. Lemma. A $c_{per}$ can be obtained from $A \in per_{\bot}$ by iterating the $'$ operation up to some ordinal $\beta$. In other terms define by transfinite induction:

$A_0 = A$

$A_{x+1} = (A_x)'$

$A_\mu = \bigcup_{x<\mu} A_x$ ($\mu$ limit ordinal).

Then for some $\beta A_\beta = A$.

Proof. Note that $A \supseteq A' \supseteq A$ and $A' = A \Rightarrow A'$ is a $c_{per}$. Q.E.D.

3.2. Lemma. (1) If $A$ and $B$ are $per_{\bot}$ then $B^A \subseteq B^A$.

(2) If $A$ is an $uper$ then $A$ is a $uper$.
The other inclusion does not hold. Preliminary recall that it is possible to give an inductive definition of TC that closes at ordinal \( \omega \), hence if \( x \in TC(R) \) then there is a finite sequence \( x \equiv w_1, w_2, \ldots, w_n \equiv y \) such that \( w_i R w_{i+1} \) for \( i = 1, \ldots, n - 1 \).

By 3.1 \( \exists \beta \) ordinal \( (A = A_\beta, B = B_\beta) \). We prove by transfinite induction on \( \gamma \) that:

\[
\phi B^A \psi \Rightarrow \phi B_\gamma^A \psi.
\]

If \( \gamma = 0 \). Trivial.

If \( \gamma = \alpha + 1 \). We have to show: \( \forall d, e \ (dA, e \Rightarrow \phi dB, \psi e) \). Suppose \( dA, e \) and consider three cases:

1. \( dA,e \). Then by ind. hyp. \( \phi dB, \psi e \) and therefore \( \phi dB, \psi e \).
2. \( (d, e) \in \text{Sup}(A_\alpha) \setminus A_\alpha \). Then \( (d, e) = d \cup X \) for some directed set \( X \) in \( A_\alpha \). In particular, \( d = \bigcup X_1 \) and \( e = \bigcup X_2 \), where \( X_1 = \{ x_1/(x_1, x_2) \in X \} \) and \( X_2 = \{ x_2/(x_1, x_2) \in X \} \) are directed in \( \text{Dom}(A_\alpha) \). By continuity \( \{(\phi x_1, \psi x_2)/(x_1, x_2) \in X \} \) is a directed set in \( B_\gamma \) and hence \( (\phi d, \psi e) = (\phi (\bigcup X_1), \psi (\bigcup X_2)) = (\bigcup (\phi X_1), \bigcup (\psi X_2)) \in \text{Sup}(B_\alpha) \subseteq B_\gamma \).
3. \( (d, e) \notin \text{Sup}(A_\alpha) \). Then there exists a finite sequence that connects \( d \) to \( e \) in \( \text{Sup}(A_\alpha) \). We get the thesis by applying (ii) to each step of the path connecting \( d \) to \( e \) and the transitivity of \( B_\gamma \).

If \( \gamma \) is a limit ordinal then the inductive hypothesis immediately applies.

We show by transfinite induction that, for each ordinal \( \gamma \), \( A_\gamma \) is uniform. The base and limit cases are trivial.

Suppose \( \gamma = \alpha + 1 \). If \( dA_\alpha, e \) then apply the ind. hyp. If \( (x, y) \in \text{Sup}(A_\alpha) \setminus A_\alpha \) then \( (x, y) = \bigcup X \), \( X \) directed in \( A_\alpha \). Let \( X_n = \{ (v_n, w_n)/(v, w) \in X \} \), observe that \( X_n \subseteq A_\alpha \), as \( A_\alpha \) is uniform by ind. hyp. and \( X_n \) is directed, being the image of a directed set via a monotone function. Therefore \( (x_n, y_n) = \bigcup X_n \in \text{Sup}(A_\alpha) \).

If \( (x, y) \in A_\alpha \setminus \text{Sup}(A_\alpha) \) then to relate the \( n \)th approximants of the extremes of the connecting sequence use the \( n \)th approximants of the intermediate elements that exist by the analysis of the previous case. Q.E.D.

3.3. Theorem. \( \text{cper}_D \) is reflective in \( \text{per}_{\perp D} \) and \( \text{uper}_{D^\infty} \) is reflective in \( \text{cuper}_{D^\infty} \).

Proof. Define \( F: \text{per}_{\perp D} \rightarrow \text{cper}_D \) as follows: \( F(A) = A \), if \( f: A \rightarrow B \) is a morphism realized by \( \phi \) then \( F(f): A \rightarrow B \) is the (unique morphism) realized by \( \phi \). Notice that this is well defined by virtue of 3.2(1). It is easy to check that indeed this is a functor. Besides Lemma 3.2(2) tells us that the restriction of \( F \) to \( \text{uper}_{D^\infty} \) gives a functor from \( \text{uper}_{D^\infty} \) to \( \text{cuper}_{D^\infty} \).
To show the existence of the natural isomorphisms it is enough to show that:

(a) \( A \perp, B \cper \Rightarrow B^4 = B^A \) (b) \( A \uplus, B \cuper \Rightarrow B^4 = B^A \).

(a) One inclusion is trivial as: \( A \supseteq A \Rightarrow B^4 \supseteq B^A \). For the other observe \( B = B \) and apply 3.2(1). (b) This is just a subcase of (a).

Q.E.D.

Having defined natural completions of \( \perp \) to \( \cper \) and \( \uplus \) to \( \cuper \) as an application, we briefly discuss the (standard) semantics of inductive types.

3.4. Definition (Inductive and recursive types). A variable \( t \) occurs positively (negatively) in a type \( x \), \( \text{Pos}(t, x) \) (\( \text{Neg}(t, x) \)), if all its free occurrences are on the left-hand side of an even (odd) number of \( \to \)'s. Add to the rules for generating types (2.1) the following clause:

\( x \) type, \( \text{Pos}(t, x) \Rightarrow \mu t \cdot x \) type.

We call the new collection of types so obtained inductive types.

For all \( \rho \equiv \mu t \cdot x(t) \) the language contains two constants, \( \text{fold}_\rho \) and \( \text{unfold}_\rho \), s.t.

\[ M : \lambda(\rho) \Rightarrow \text{fold}_\rho M : \rho \quad \text{and} \quad M : \rho \Rightarrow \text{unfold}_\rho M : \lambda(\rho), \]

whose computational behavior is determined by the equations:

\[ \text{fold}_\rho (\text{unfold}_\rho x) = x \quad \text{and} \quad \text{unfold}_\rho (\text{fold}_\rho x) = x. \]

Recursive types are simply obtained by dropping the condition \( \text{Pos}(t, x) \) in the previous clause.

The study of inductive definitions over the structures of arithmetic and analysis was also started by Kleene and continued by Spector. Generalizations of the main results to a more abstract framework were developed in the 70's (see, e.g., Moschovakis, 1974). A variety of structures in mathematics and computer science (e.g., natural numbers, lists, trees) admit an inductive definition. Mendler (1988) recently reconsidered the problem in the framework of type theory. He proved that inductive types are exactly the recursive types that, when added to the pure second-order \( \lambda \)-calculus, preserve the property of strong normalization of the typable \( \lambda \)-terms.

Let us notice that in general if \( C \) is one of the subcategories of per introduced in Section 1 then (the collection of objects of) \( C \) is a complete lattice w.r.t. the partial order given by set-theoretic containment (\( C \) has arbitrary glb's, i.e., intersections, and hence arbitrary lub's). However, the property of completeness is not preserved under arbitrary unions, therefore the lub's are actually computed as completions of set-theoretic unions.
3.5. DEFINITION. Let \( \Gamma \) be a monotone operator over \( C \). Define by transfinite induction:

\[
\begin{align*}
\Gamma_0 &= \emptyset \cap C \\
\Gamma_{\alpha+1} &= \Gamma(\Gamma_\alpha) \\
\Gamma_\lambda &= \emptyset \cup \{ \Gamma_\alpha / \alpha < \lambda \} \text{ for } \lambda \text{ limit ordinal (check that this is always well defined).}
\end{align*}
\]

3.6. PROPOSITION. Let \( \Gamma \) be a monotone operator over \( C \), then there is a least \( \alpha \) s.t. \( \Gamma_{\alpha + 1} = \Gamma_\alpha \) and \( \Gamma_\alpha \) is the least fixed point (lfp) of \( \Gamma \).

Proof. The existence of such ordinal \( \alpha \) is shown by an obvious cardinality argument. Besides suppose \( \Gamma A = A \) then observe \( A \supseteq \Gamma_\gamma \) by transfinite induction over the ordinal \( \gamma \). Q.E.D.

We wish to extend the type-interpretation in 2.4 by setting

\[
[\mu t \cdot \alpha]_\eta = \text{lfp}(\lambda A \cdot [\alpha]_{\eta[A/\eta]}).
\]

A rigourous treatment requires the introduction of a new interpretation:

\[
[\ ]': \text{Rec Type} \rightarrow (Tenv \rightarrow (C \cup \{ \uparrow \})),
\]

where \( \uparrow \) stands for undefined and \( \text{Rec Type} \) are the recursive types. The idea is that \([ \ ]'\) coincides with \([ \ ]\) on Type (see 2.4) and interprets recursive types according to clause (\(*\)) whenever this makes sense. It is intended that the interpretation of a type is defined if all the interpretations of its subtypes are defined.

It is an obvious exercise to complete the following definition:

\[
[\alpha \rightarrow \beta]_\eta = \text{lfp}(\lambda \beta \cdot [\beta]_{\eta[A/\eta]}), \quad \text{if both } \alpha \text{ and } \beta \text{ interpretations are defined, } \uparrow.
\]

\[
[\mu t \cdot \alpha]_\eta = \text{lfp}(\Gamma) \quad \text{if } \Gamma \equiv \lambda A \cdot [\alpha]_{\eta[A/\eta]} \text{ is monotone and } [\alpha]' \text{ is defined, } \uparrow.
\]

Eventually we need the following:

3.7. PROPOSITION. \([ \ ]'\) extends \([ \ ]\) on \( T_\rho \) and it is defined for every inductive type.

Proof. A standard induction on the structure of \( \alpha \). Q.E.D.

Notes. (1) We have given a standard interpretation of inductive types, in Section 4 we will show that working in cuper it can be extended to an interpretation of recursive types.
(2) The study of the relationships between second-order and inductive definitions lies outside the scope of this paper. Nevertheless it appears that the natural setting for this study are the non-pointed categories, as in the other cases the relationship between propositions and (data-)types is definitely lost; e.g., there is no relation in \( \mathcal{C}(u) \) per between \( \forall t \cdot (\alpha(t) \to t) \to t \) and \( \mu t \cdot \alpha(t) \), where \( \text{Pos}(t, \alpha(t)) \) (recall that if \( \alpha(t) \) is a first-order formula of arithmetic then the previous formulae "define" the same structure).

4. Contractive Operators and Recursive Types

It is a well known fact that truly recursive definitions of data types are relevant in the practice of programming languages' theory (e.g., denotational semantics). Suppose we are interested in a \( \mathcal{C}(u) \) per \( A \) s.t. \( A \cong B^A \). Taking a set-theoretic approach one could try to express \( A \) as the "limit" of a sequence of \( \mathcal{C}(u) \) pers \( \{A_n\}_{n<\omega} \) iteratively defined as \( A_{n+1} = B^{A_n} \). It is readily realized that the antimonotonicity, in its first argument, of the exponentiation operator does not lead to any conclusive result about the asymptotic behaviour of the sequence (as for a categorical point of view remember that the exponent functor is contravariant anyway).

Still there is a chance that the succession is convergent with respect to some metric over the space of \( \mathcal{C}(u) \) per's. This idea is developed in MacQueen et al. (1986) w.r.t. to the space of ideals, that can be characterized as \( \mathcal{C}(u) \) per's closed downward, hence with just one equivalence class (this is not a model for the second-order \( \lambda \)-calculus though, see Cartwright, 1984; Martini, 1988, for other solutions, as yet, unrelated to realizability models).

A notion of closeness is introduced that exploits the stepwise construction of the underlying \( D_\infty \) \( \lambda \)-model. Roughly speaking the closeness of two ideals is the greatest stage of the construction (if there is one) at which they still coincide.

Starting from this observation an (ultra-)metric is obtained for which the space is complete. The next obvious step is to check if by chance the operators we are considering are contractive as this would lead to a (unique) solution of fix-point equations by means of Banach's theorem (Banach, 1922).

In extending this approach to \( \mathcal{C}(u) \) per's the following problem arises: two \( \mathcal{C}(u) \) per's can coincide at every stage of the construction of \( D_\infty \) but wildly differ at the limit (this is not the case for ideals because of the downward closeness). Therefore we are naturally led to add a uniformity condition, namely: \( dAe \Rightarrow \forall n \, d_nAe_n \), i.e., the category of \( \mathcal{C}(u) \) per's.

For people familiar with MacQueen et al. (1986), we emphasize that our metric refers to the approximating elements rather then the finite ones, hence we can work in arbitrary cpo's. On the other hand, we just consider
a metric naturally suggested by the $D_\infty$ construction rather than an arbitrary one generated by a "rank map." These two choices entail, we hope, a synthetic and elementary exposition.

Last but not least we would like to comment on the question: what have metric spaces got to do with realizability structures? Probably nothing. We are just using certain familiar notions from the theory of metric spaces to show the convergence of certain sequences to a limit that has, we think, a clear computational intuition (see 5.6).

4.1. **Definition** (Closeness). We define a function $c: \text{cuper}^2 \to \omega \cup \{\infty}\). Assume, for $A$ cuper, $A_{|n} \equiv_{abst} A \cap (i_n(D_n))^2$ (see 1.1; henceforth we ambiguously denote $D_n$ and $i_n(D_n)$ with $D_n$). If $A \neq B$ then $c(A, B) = \max\{n/A_{|n} = B_{|n}\}$ and $c(A, B) = \infty$ o.w.

4.2. **Proposition.** (1) The definition is correct as given $A, B$ cuper's: $A = B \iff \forall n A_{|n} = B_{|n}$.

(2) $\forall A, B, C$ cuper's $c(A, C) \geq \min\{c(A, B), c(B, C)\}$.

**Proof.** (1) Immediate. (2) Observe: $A_{|n} = B_{|n}$ and $B_{|m} = C_{|m} \Rightarrow A_{|k} = C_{|k}$, where $k = \min\{n, m\}$. Q.E.D.

4.3. **Definition** (Distance). $d: \text{cuper}^2 \to \mathbb{R}^+$. If $A \neq B$ then $d(A, B) = \exp(2, -c(A, B))$ and $d(A, B) = 0$ o.w.

4.4. **Theorem.** (1) $(\text{cuper}, d)$ is a metric space, indeed an ultra-metric.

(2) It is complete.

**Proof.** (1) The only remarkable point is that a stronger form of the triangular inequality can be proved, namely: $\forall A, B, C \ d(A, C) \leq \max\{d(A, B), d(B, C)\}$. This is immediate from 4.2 (2).

**Note.** This kind of metrics also arise when dealing with spaces of infinite labelled trees (Arnold and Nivat, 1980). They are compact if, roughly speaking, the collection of distinct objects up to the $n$th stage is finite (like in the case $At$ is finite, see 1.1).

(2) Let $\{A_n\}_{n<\omega}$ be a Cauchy sequence, i.e., $\forall \varepsilon > 0, \exists n_\varepsilon, \forall n, m \geq n_\varepsilon d(A_n, A_m) < \varepsilon$. We will build $A = \lim\{A_n\}_{n<\omega}$ by stages. Note that: $\forall N > 0, \exists n(N), \forall n \geq n(N), A_{|nN}$ is constant.

Let $A_{|N} = (A_{|n(N)})_{N \in \omega}$. $\{A_{|N}\}_{N<\omega}$ is a chain of cuper's. Let $B = \cup \{A_{|N}\}_{N<\omega}$ and define $A$ as the completion of the upper $B$ to a cuper, i.e., $A = B$. To prove that indeed $A$ is the limit it is enough to show that $\forall k, A_{|k} = A_{|k};$ i.e., the operation of completion does not add new approximating elements to $B$. Note that this is false for a generic upper but
4.5. Note. Let \((X, d)\) be a metric space. \(\Gamma: X \to X\) is contractive (and hence uniformly continuous) if there is a \(\kappa\) s.t. \(0 < \kappa < 1\) and \(\forall x, y \in X, d(\Gamma(x), \Gamma(y)) \leqslant \kappa d(x, y)\). \(\Gamma\) is also said non-expansive if \(\forall x, y \in X, d(\Gamma(x), \Gamma(y)) \leqslant d(x, y)\). Banach’s theorem states that a contractive operator \(\Gamma\) over a complete metric space admits a unique fixed point that can be “computed” as the limit of the Cauchy succession \(\{\Gamma^n(x)\}_{n < \omega}\) for an arbitrary \(x \in X\).

In our model the space of type operators coincides with that of all set-theoretic functions (1.6.1); therefore the only possibility of applying Banach’s theorem comes from looking at a restricted class of definable operators.

Assume the following natural distances on (check left to the reader; see also 4.9):

- **Cartesian product:** \(\forall A, A', B, B'\) cuper’s, \(d((A, B), (A', B')) = \max\{d(A, A'), d(B, B')\}\);
- **Functional space:** \(\forall F, G\) cuper \(\to\) cuper, \(d(F, G) = \max\{d(F(A), G(A))|A\text{ cuper}\}\).

Then the following proposition shows that the exponentiation is contractive and the indexed product (or intersection) is nonexpansive.

4.6. **Proposition.** \(\forall A, A', B, B'\) cuper’s, \(\forall F, G\) cuper \(\to\) cuper,

\[
(1) \quad d(B^A, B'^A) \leqslant (1/2) d((A, B), (A', B'))
\]

\[
(2) \quad d(\prod (F), \prod (G)) \leqslant d(F, G).
\]

**Proof.** (1) Note that: \(A_{jk} = A'_{jk}\) and \(B_{jk} = B'_{jk} \Rightarrow B^A_{jk+1} = B'^A_{jk+1}\).

(2) Observe: \(\cap \{F(A)/A\text{ cuper}\}_{jk} \neq \cap \{G(A)/A\text{ cuper}\}_{jk} \Rightarrow \exists A, F(A)_{jk} \neq G(A)_{jk}\) Q.E.D.

Now we set

\[ [\mu t \cdot x]_\eta = \text{lfp}(\lambda A \cdot [x]_{\eta[A/t]}) \quad (\text{lfp} \equiv \text{abr. least fixed point}).\]

Rigorously speaking we change the algebra of polymorphic types (see 1.6.1) so that \([T \to T]\) is the collection of contractive maps plus the identity and we extend the algebra by requiring a fixed point operator \(\text{fix}: [T \to T] \to T\). Of course we have to check that \(\forall \eta, \lambda A \cdot [x]_{\eta[A/t]} \subseteq [T \to T];\) this is exactly the contents of the following lemma.

As for terms’ interpretation, since we are solving recursive type equations
up-to-equality, the constants fold_{μ·x} and unfold_{μ·x} (see 3.4) will be just the identity morphism on the interpretation of μ·x.

Also note that similar results of contraction can be obtained for other type constructors like product and sum, modulo a suitable construction of the $D_\omega$ model (the reader can check that in our case the product is contractive).

4.7. Lemma. Denote with $(A_n)$ a n-tuple $A_1, ..., A_n$ of cuper's and with $(t_n)$ a n-tuple $t_1, ..., t_n$ of type variables. Then ∀α type, ∀n ∈ ω, ∀η type environment:

1. $d([x_1 \to x_2]_{η[(A_1)/(t_1)]}, [x_1 \to x_2]_{η[(B_1)/(t_1)]}) = d([x_2]_{η[(A_1)/(t_1)]}, [x_2]_{η[(B_1)/(t_1)]})$.
2. The operator $\lambda A \cdot [x]_{η[A/A]}$ is either contractive or the identity.

Proof. We proceed by combined induction on the structure of α.

α is a variable. Trivial case analysis.

α ≡ α_1 → α_2.

1. $d([x_1 \to x_2]_{η[(A_1)/(t_1)]}, [x_1 \to x_2]_{η[(B_1)/(t_1)]}) = d([x_2]_{η[(A_1)/(t_1)]}, [x_2]_{η[(B_1)/(t_1)]})$.
2. By 4.6 and (1) in this case the operator is always contractive.

α 3 t's = fl. The only interesting case is if s ≠ t.

1. $d([\forall s \cdot \beta]_{η[(A_1)/(t_1)]}, [\forall s \cdot \beta]_{η[(B_1)/(t_1)]}) = d(\exists A \cdot [\beta]_{η[(A_1)/(t_1)]}, η[(B_1)/(t_1)]), \exists A \cdot [\beta]_{η[(B_1)/(t_1)]}, \exists A \cdot [\beta]_{η[(B_1)/(t_1)]})$. Apply ind. hyp. and 4.6(2).

2. Assume $t ≡ FV(β)$ (o.w. it is trivial). We need some "look-ahead": either $β ≡ Q_t_1 ... Q_t_n \cdot t$ ($Q ::= ∀ | μ$) and the operator is the identity or $β ≡ Q_t_1 ... Q_t_n \cdot β_1 → β_2$ and the operator is contractive.

α = μs · β. We just analyse the case: s ≠ t.

1. $d([μs \cdot β]_{η[(A_1)/(t_1)]}, [μs \cdot β]_{η[(B_1)/(t_1)]}) = d(Lfp(Γ_1), Lfp(Γ_2))$, where: $Γ_1 = A \cdot [β]_{η[(A_1)/(t_1)]}$ and $Γ_2 = A \cdot [β]_{η[(B_1)/(t_1)]}$. Apply ind. hyp. and case analysis, either both the identity or both contractive. Consider the second case and recall that $Lfp(Γ_j) = \lim_{n < ω} Γ^n_j(⊥)$. Prove, using ind. hyp. on β, that: ∀n, $d(Γ^n_1(⊥), Γ^n_2(⊥)) ≤ d((A_n), (B_n))$; the thesis immediately follows.

2. The operator is: $\lambda A \cdot Lfp(λB \cdot [β]_{η[A/A \cdot B/B]}$). Interesting case:

$Γ_j = ...$, $s ∈ FV(β)$ and $λB \cdot [β]_{η[A/A \cdot B/B]}$ contractive. Use again the characterization of the Lfp.

Q.E.D.

4.8. Remarks. (1) Over inductive types and in the category of cuper's this interpretation coincides with that given in Section 3 as both interpret a type $μ·x$ as the least fixed point of the related operator: $\lambda A \cdot [x]_{η[A/A]}$. 


In particular in cuper the closure ordinal of such an operator is at most \( \omega \) (it is easy to prove this fact exploiting 5.6).

2) The models obtained using the categories of cper's and cuper's are quite different. Consider, e.g., the type \( \forall t \cdot t \rightarrow t \). In cper its interpretation contains only two elements, namely \( \perp \) and identity. In cuper there are an infinity of "non-standard" elements, e.g., the previous ones and all the \( n \)th approximations of the identity since \( \text{id}_n [\forall t \cdot t \rightarrow t] \) \( \text{id}_n \) iff \( \forall A, \forall d, e, dA = \text{id}_n dA \text{id}_n e \) that holds since \( A \) is a cuper and \( \text{id}_n e = e_{n-1} \).

3) The interpretation over cper of pure second-order types is contained in that over cuper for all closed \( \prod_1 \) types. A closed \( \prod_1 \) type \( \alpha \) is always equal to one of the shape \( \forall t_1 \cdot \ldots \cdot \forall t_n \cdot \alpha_1 \rightarrow \ldots \rightarrow \alpha_m \rightarrow t_i \) (\( \alpha_j \) quantifier free for \( j = 1, \ldots, m \)). Hence observe that: \( [\alpha]^{\text{cper}} = \bigcap A_1 \ldots \text{Anuper} F(A_1, ..., An) \) and \( [\alpha]^{\text{cuper}} = \bigcap A_1 \ldots \text{Anuper} F(A_1, ..., An) \), where \( F \) is an operator on \( \mathcal{U} \) per's defined as \( F(A_1, ..., An) = \Delta [\alpha_1 \rightarrow \ldots \rightarrow \alpha_m \rightarrow t_i]_{\{A_1/1, ..., An/m\}} \). In general this property fails as the arrow is contravariant; e.g., compute the interpretations of \( r \rightarrow r \) where \( r = t \rightarrow t \).

4) Types with the shape \( \mu t \cdot \sigma_1 \cdot \ldots \cdot \sigma_m t, \ t \in \mathcal{F}V(\sigma_1 \cdot \ldots \cdot \sigma_m t) \), where \( \sigma_j ::= \alpha_i \rightarrow |s| \mu s (s \neq t) \) are terminal objects in this interpretation (e.g., \( \mu t, t \rightarrow t \)). Proof hint: use \( 1^t \cong 1 \) (1 terminal object), indexed product of terminal objects is terminal and the construction of the fixed point.

4.9. Higher Order Domain Equations

Consider the full set-theoretic functional hierarchy over the space (cuper, \( d \)). There is a very simple way to inherit the structure of complete metric space. Namely, inductively define the distance \( d_{K ightarrow H} \) between two functionals \( F, G \) from the space \( K \) to the space \( H \) as follows:

\[
d_{K ightarrow H}(F, G) = \Delta \ max\{d_H(F(L), G(L)) | L \text{ functional in } K\}.
\]

Of course, a simply typed \( \lambda \)-calculus enriched with type constructors \( \forall \) and \( \rightarrow \) (i.e., the type constructors of Girard's system \( \mathcal{F}^\omega \)) can be interpreted in this set-theoretic structure.

The problem that remains to be solved is that of the characterization of the contractive, definable operators. This seems a nontrivial task as in this case the language of type constructors includes higher order features.

5. Another Approach to Recursive Types

In this section we will adapt to our context and generalize a solution to the problem of interpreting recursive types in realizability structures due to Mario Coppo (Coppo, 1985; Coppo and Zacchi, 1986). This approach is
more elementary as it does not require the introduction of topological
notions and it gives an additional insight into the construction of the limit
(used in 6.3) but indeed it is our personal experience that it is harder to
grasp and explain as we have to rely on merely combinatorial notions.

By a more careful analysis we are able to eliminate a syntactic restriction
on the set of interpretable types in Coppo and Zacchi, who exclude the
"non-contractive" ones. Similar results have been obtained independently
and at about the same time by Felice Cardone (1988) by means of a
slightly different technique.

5.1. DEFINITION (Rank). The rank of a second-order recursive type is
given by the number of the external quantifiers \( \forall \) and \( \mu \). That is, if
\( \alpha \equiv \bigwedge_{1}^{n} \bigwedge_{1}^{m} \bigcap_{1}^{p} \bigvee_{1}^{q} \beta \), where \( n \geq 0 \), \( Q \in \{ \forall, \mu \} \), and \( \beta \) does not begin with a
quantifier then \( \text{rank}(\alpha) = n \).

The idea is to build the solution of the fixed point equation as the "limit"
of a certain growing succession of "approximations."

5.2. DEFINITION. We define a function \( R: \omega \times T_{p} \times T_{\text{env}} \rightarrow \text{cuper} \)
\( (T_{\text{env}} = T_{\text{var}} \rightarrow \text{cuper}) \) by induction on the pair \( \langle n, \text{rank}(\alpha) \rangle \) (with the usual
lecicographic order), \( \alpha \) type, \( \eta \) type environment:

\[
\begin{align*}
(0) \quad R(0, \alpha, \eta) &= \{(\perp, \perp)\} \\
(\text{var}) \quad R(n + 1, t, \eta) &= \eta(t) \cap D_{n+1} \times D_{n+1} \\
(\rightarrow) \quad R(n + 1, \alpha \rightarrow \beta, \eta) &= R(n, \beta, \eta) \cap D_{n+1} \times D_{n+1} \\
(\forall) \quad R(n + 1, \forall \cdot \alpha, \eta) &= \bigcap_{\alpha \in \text{cuper}} R(n + 1, \alpha, \eta[A[/t]] \\
(\mu) \quad R(n + 1, \mu \cdot \alpha, \eta) &= R(n + 1, \alpha, \eta[R(n, \mu \cdot \alpha, \eta)/t]).
\end{align*}
\]

Note. By inspecting the definition is readily realized that \( R(n, \alpha, \eta) \) is a
cuper in \( D_{n} \times D_{n} \), since cuper is closed under arbitrary intersections and
\( \{(\perp, \perp)\}, \ D_{n} \times D_{n} \) are cupers. Observe that in the "potentially circular"
clauses \( \forall \) and \( \mu \) the rank of the type decreases. The second tricky point
consists in observing that although the exponentiation is a antimonotone
\( \{R(n, \alpha, \eta)/n < \omega\} \) is a growing sequence as shown by the following lemma.

5.3. BASIC LEMMA. For all \( n, \alpha, \eta \): \( R(n, \alpha, \eta) = R(n + 1, \alpha, \eta) \cap D_{n} \times D_{n} \).

Proof. By induction on \( \langle n, \text{rank}(\alpha) \rangle \).

\( n = 0 \). Trivial, \( \{(\perp, \perp)\} \) is the least cuper w.r.t. set-theoretic contain-
ment.

\( n = m + 1 \). Proceed by case analysis on \( \alpha \).

(\text{var}) Just recall that: \( D_{n+1} \supseteq D_{n} \).
We will show the two inclusions. Suppose $(f, g) \in R(n + 1, \alpha \to \beta, \eta) \cap D_n \times D_n$. Then $fR(n, \alpha \to \beta, \eta) g$, since: $dR(n - 1, \alpha, \eta) e \Rightarrow \text{ind. hyp.}$ $dR(n, \alpha, \eta) e \Rightarrow \text{def. } fdR(n, \beta, \eta) ge \Rightarrow \text{ind. hyp.}$ $fdR(n - 1, \beta, \eta) ge$ (note that, e.g., $fd \in D_{n-1}$).

Vice versa, let $(f, g) \in R(n, \alpha \to \beta, \eta)$. Then $(f, g) \in D_n \times D_n$ and $fR(n + 1, \alpha \to \beta, \eta) g$, since: $dR(n, \alpha, \eta) e \Rightarrow \text{def. } d_{n-1}R(n, \alpha, \eta) e_{n-1} \Rightarrow \text{ind. hyp.}$

$R(n, \alpha, \eta) e_{n-1} \Rightarrow \text{def. } fd_{n-1}R(n - 1, \beta, \eta) ge_{n-1} \equiv fdR(n - 1, \beta, \eta) ge$.

Observe that the two expressions do not quite match since by ind. hyp. we just have: $R(n - 1, \mu \cdot \alpha, \eta) = R(n, \mu \cdot \alpha, \eta) \cap D_{n-1} \times D_{n-1}$. Therefore, we need some further analysis on the structure of $\alpha$:

If $t \notin FV(\alpha)$ then $A = B$, since clearly $R(n, \alpha, \eta)$ depends on $\eta$ only w.r.t. the value of the variables free in $\alpha$.

If $t \in FV(\alpha)$ and $\alpha \equiv Q_1, t_1 \cdots Q_n, t_n \beta \rightarrow \gamma$ ($n \geq 0$), then when we evaluate the relation corresponding to the variable $t$ we are only interested in the elements in $D_{n-1} \times D_{n-1}$ (the $(\rightarrow)$ clause decreases $n$).

If $t \in FV(\alpha)$ and $\alpha \equiv Q_1, t_1 \cdots Q_n, t_n$ ($n \geq 0$), then, in general, $R(n, \alpha, \eta) = \eta(t) \cap D_n \times D_n$.

Hence $A = R(n, \mu \cdot \alpha, \eta) = R(n + 1, \alpha, \eta[R(n, \mu \cdot \alpha, \eta)/t]) \cap D_n \times D_n = B$.

Q.E.D.

5.4. **Definition.** Define $R : Tp \times T_{env} \rightarrow \text{uper}$ as follows: $dR(\alpha, \eta) e$ iff $\forall n < \omega, d_n R(n, \alpha, \eta) e_n$, and set $[\mu \cdot \alpha]_p = \text{def. } R(\mu \cdot \alpha, \eta)$.

The following theorem says that the previous definition makes sense and that it is an extension of the intended interpretation.

5.5. **Theorem.** (1) $R(\alpha, \eta)$ is the completion of the uper: $\bigcup_{n < \omega} R(n, \alpha, \eta)$.

(2) Besides:

$$R(\alpha \rightarrow \beta, \eta) = R(\beta, \eta)^{R(\alpha, \eta)}$$

$$R(\forall t \cdot \alpha, \eta) = \bigcap_{A_{\text{uper}}} R(\alpha, \eta[A/t])$$

$$R(\mu \cdot \alpha, \eta) = R([\mu \cdot \alpha/t] \alpha, \eta).$$
Proof. (1) We just mention the main steps: (a) \( R(\alpha, \eta) \) is a per. (b) \( \forall n, R(\alpha, \eta) \supseteq R(n, \alpha, \eta) \), since \( dR(n, \alpha, \eta) e \Rightarrow \forall m, d_m R(n, \alpha, \eta) e_m \). (c) By definition, \( R(\alpha, \eta) \) is uniform. (d) Completeness easily follows by the continuity property of \( D \). (e) Clearly each cuper containing \( R = \bigcup_{n<\omega} R(n, \alpha, \eta) \) has to contain \( R(\alpha, \eta) \) as every element in this cuper is the sup of a directed set in \( R \) (see proof of Theorem 4.4).

(2) \( \forall \alpha \in \beta \) We show the two containments. Suppose \( fR(\alpha + \beta, \eta) g \) and \( dR(\alpha, \eta) e \). Recall that: \( fd = \bigcup_{n<\omega} f_{n+1} d_n \). Therefore it is sufficient to prove: \( \forall n, f_{n+1} d_n R(n, \beta, \eta) g_{n+1} e_n \) that trivially follows by a rewriting of the hypothesis. Vice versa, suppose \( fR(\beta, \eta) R(\alpha, \eta) g \) and \( dR(n, \alpha, \eta) e \). Then \( dR(n, \alpha, \eta) e \Rightarrow dR(\alpha, \eta) e \Rightarrow fdR(\beta, \eta) ge \Rightarrow (fd_n) R(n, \beta, \eta)(ge_n) = f_{n+1} dR(n, \beta, \eta) g_{n+1} e_n \).

(\forall) \( fR(\forall \alpha, \eta) g \) iff \( \alpha \forall n, f_n R(n, \alpha, \eta) g_n \) iff \( \forall \alpha \) cuper \( \forall n, f_n R(n, \alpha, \eta) g_n \) iff \( \forall \alpha \) cuper \( fR(\alpha, \eta) g \).

(\mu) \( fR(\mu \alpha, \eta) g \) iff \( \forall n, f_n R(n, \mu \alpha, \eta) g_n \) iff \( \forall n, f_n R(n, \mu \alpha, \eta) g_n \) iff \( \forall n, f_n R(n, \mu \alpha, \eta) g_n \) iff \( \forall n, f_n R(n, \mu \alpha, \eta) g_n \) iff \( \forall n, f_n R(n, \mu \alpha, \eta) g_n \) iff \( fR(\mu \alpha) g \).

If \( t \notin FV(\alpha) \): trivial. If \( t \in FV(\alpha) \) and \( \alpha \equiv Q_1 t_1 \cdots Q_n t_n \ (n \geq 0) \), then

\[
R(n, \alpha[N \in \epsilon] = R(n-1, \mu \alpha[N \in \epsilon]) = R(n, \mu \alpha[N \in \epsilon]) = R(n, \alpha[N \in \epsilon])
\]

If \( t \in FV(\alpha) \) and \( \alpha \equiv Q_1 t_1 \cdots Q_n t_n \beta \rightarrow \gamma \ (n \geq 0) \), observe that when we evaluate the relation corresponding to the variable \( t \) we are only interested in the elements in \( D_{n-1} \times D_{n-1} \).

The additional insight given by this technique is summarized in the following proposition saying that the \( n \)th iterates of a definable operator \( \Gamma \) cut at the \( n \)th level are a growing sequence. It immediately follows that the interpretation of recursive types just given coincides with that in the previous paragraph.

5.6. PROPOSITION. For all \( t \) type variables \( \alpha \) type, \( \eta \) type environment let \( \Gamma_{t, \alpha, \eta} \equiv abr \), \( \overline{\Lambda} \), \( \overline{\Lambda} \) and \( \Gamma^n \equiv abr \). \( \Gamma \circ \cdots \circ \Gamma, \ n \) times. Then, \( R(n, \mu \alpha, \eta) = \Gamma^n (\perp) \cap D_n^3 \).

Proof. Proceed by induction on \( n \); by now it is a simple exercise.

Q.E.D.

As in the lemma we need some analysis on the structure of \( \alpha \).
RECURSION OVER REALIZABILITY STRUCTURES

6. SOME LOCAL PROPERTIES OF MODELS

In this paragraph some results about term equations, type isomorphisms, and type equalities that hold in the models introduced are presented.

We assume that $\text{Fix}$ (see 2.3) is $\lambda$-definable by a fixed-point combinator $Y$. E.g., in continuous $\lambda$-models (see Barendregt, 1984) every fixed-point combinator defines $\text{Fix}$.

6.1. Term Equations

6.1.1. DEFINITION. Define a function erasure $\text{er}: \text{Typed-terms} \rightarrow \text{Type-free-terms} \cup \{\varepsilon\}$ ($\varepsilon$ empty string) by induction on $M$: $\text{er}(MN) = \text{er}(M) \text{er}(N)$; $\text{er}(MX) = \text{er}(M)$; $\text{er}(\lambda x^\alpha \cdot M) = \lambda x \cdot \text{er}(M)$; $\text{er}(\lambda t \cdot M) = \text{er}(M)$; $\text{er}(YM) = Y \text{er}(M)$ ($Y$ fixed-point combinator). $\text{er}($fold$_{\mu \cdot x}) = \text{er}($unfold$_{\mu \cdot x}) = \varepsilon$.

6.1.2. DEFINITION (Compatible environments). Let $\eta: \text{Type Var} \rightarrow \text{Ob}_C$, $\rho: \text{Typed Var} \rightarrow \bigcup_{\alpha \in \text{Ob}_C} D/A$, $\xi: \text{Var} \rightarrow D$, where we assume that for all typed variables $x^\alpha$, $y^\beta$ if $\alpha \neq \beta$ then $x \neq y$. We say that $(\eta, \rho, \xi)$ are compatible iff

$$\forall x^\alpha \in \text{Typed Var}, \rho(x^\alpha) \in D/[\alpha]_\eta \quad \text{and} \quad \xi(x) \in \rho(x^\alpha).$$

The following precisely defines the relation between the interpretations of a typed term and its erasure. It extends a result in Mitchell (1986) concerning the pure second-order $\lambda$-calculus.

6.1.3. THEOREM. Given $M, N: \alpha$, denote with $P^\alpha$ the $\lambda \perp$-term corresponding to the $\text{BT}(P)$ ($\text{BT}$ for Bohm tree) cut at the $n$th level. Then for all $\eta, \rho, \xi$ compatible the following holds:

1. $[M]_\rho^\eta = [\text{er}(M)]^\xi_{\xi [\alpha]_\eta \in C}$. Besides if $D$ is a continuous $\lambda$-model then:
   2. $\text{BT}(\text{er}(M)) = \text{BT}(\text{er}(N)) \Rightarrow C \models M = N$.
   3. $[M]_\rho^\eta = \bigcup_{n \in \omega} [\text{er}(M)^{\alpha_n}]^\beta_{\beta [\alpha]_\eta \in C}$.

Proof: (1) By induction on the structure of $M$. The thesis is equivalent to: $[\text{er}(M)]^\xi_{\xi [\alpha]_\eta \in C} \in [M]_\rho^\eta$.

$M \equiv x^\alpha$. Immediate, by the definition of compatible environment.

$M \equiv \lambda x^\cdot \cdot N^\beta$. Consider the term interpretation. Let $d[\alpha], d$ then $[\lambda x \cdot \text{er}(N^\beta)]^\xi_{\xi [\alpha]_\eta \in C} = [\text{er}(N^\beta)]^\xi_{\xi [\alpha]_\eta \in C}$ by inductive hypothesis.
$M \equiv N^{a \rightarrow \beta} P^a$. If $N$ is a constant see following cases, o.w.: $[\text{er}(N^{a \rightarrow \beta} P^a)]_\xi = [\text{er}(N^{a \rightarrow \beta})]_\xi \cdot [\text{er}(P^a)]_\xi \in [N^{a \rightarrow \beta} P^a]_{P\eta}$ by inductive hypothesis on $N$ and $P$.

$M \equiv \lambda t \cdot M^t$ and $M \equiv M^{\forall t \cdot \beta}$. Analogous to the previous cases.

The following two cases make sense respectively in c(u)per and in cuper:

$M \equiv YN^{a \rightarrow \alpha} \cdot \text{er}(YN^{a \rightarrow \alpha})]_\xi = \text{Fix}[\text{er}(N^{a \rightarrow \alpha})]_\xi \in \text{Fix}_{\{\alpha\}} [N^{a \rightarrow \alpha}]_{P\eta}$.

$M \equiv \text{fold } N$ or $M \equiv \text{unfold } N$. Recursive equations are solved up to equality.

(2) A continuous $\lambda$-model equates terms with the same Böhm trees, by (1) we can conclude that the interpretation of the typed terms is the same.

(3) We have just rewritten (1) using the approximation property of $D$. Q.E.D.

Remarks. (1) One good reason to choose a $\lambda$-model rather than a pca in the model constructions previously given, is that there is a very simple way to compute a syntactic expression (i.e., the erasure) whose denotation is a realizer for the interpretation of the typed term.

Another important aspect is the possibility of giving a unified semantic treatment of types a'la Church (explicit typing) and a'la Curry (type-inference). Indeed per structures also provide models for type free languages with a type assignment system via the so-called quotient-set semantics (Scott, 1976).

(2) Theorem 6.1.3. gives only a sufficient condition to equate typed terms. E.g., the following equation holds: $(\text{cu})\text{per} \models \lambda^x \forall t \cdot x^{\forall t \cdot t} \cdot (\forall t \cdot t) = \lambda^x \forall t \cdot x^{\forall t \cdot t}$. Nevertheless sound equations like: $(\text{c})\text{per} \models \lambda^x \forall t \cdot x^{\forall t \cdot t}$, $\lambda^y \forall t \cdot t \cdot (\forall t \cdot t) \cdot (\forall t \cdot t) = \lambda^x \forall t \cdot x^{\forall t \cdot t} \cdot \forall t \cdot t$ require a finer analysis. The point is that terms that can be distinguished in a type free discipline can very well be equated in a typed one. E.g., recall that the erasure of a pure second-order $\lambda$-term is strongly normalizable hence by Bohm's theorem no two distinct such terms can be consistently equated in a $\lambda$-model, whereas this can be the case in the typed model as shown by the previous equation.

6.1.4. Excursus. In 6.1.3 we give a weak form of the approximation theorem, as we have to make a detour through the approximations of the erasure of the typed term. Indeed we have not defined a complete partial order on the quotient space $D/A$, for $A \text{ cper}$, so it is not even possible to formulate the classical theorem of the type-free case. We are going to define another subcategory of per's (lcper), it will be useful to suggest two ideas:

(i) There are many more conditions that can be imposed on per's and that still lead to full cartesian closed subcategories, in this sense the
situation is analogous to that arising in domain theory where a number of interesting subcategories of cpo's have been considered.

(ii) It seems relevant to order the quotient space $D/A$; this could lead to a nicer formulation of the approximation theorem and more interestingly to an $O$-category (Wand, 1979; Smyth and Plotkin, 1982).

**DEFINITION.** Suppose that $D$, our $\lambda$-model in 1.1, is a complete lattice and $D \times D \cong D$. Say that $A$ is a leper (1 for lattice) iff

(i) $A$ is a cper

(ii) every equivalence class in $D/A$ is a “full” complete sublattice, i.e.,

$$dAd \Rightarrow \bigcap [d]_A A \cup [d]_A \quad \text{and} \quad d \leq e \leq f, dAf \Rightarrow dAe.$$

In other words an equivalence class is completely determined by its largest and least elements. Also observe that if $a, b \in D/A$ then $a = b$ iff $\bigcup a \leq \bigcup b$; $a \leq b$ iff $(\bigcap a) \cap (\bigcup b) \geq (\bigcap a) \cup (\bigcup b)$. The morphisms are still like in Section 1.

**PROPOSITION.** leper is a ccc and it is closed under arbitrary set theoretic intersections.

**Proof.** Left to the reader, see 1.4. Q.E.D.

It is now possible to define various, simple partial orders on the quotient spaces, e.g., let $A$ be a leper, $a, b \in D/A$: (1) $a \leq b$ iff $\bigcup a \leq \bigcup b$; (2) $a \leq b$ iff $\bigcap a \leq \bigcap b$; (3) $a \leq b$ iff $(\bigcap a, \bigcup a) \leq ((\bigcap b, \bigcup b)$. Nevertheless none of these partial orders has satisfying completeness properties or behaves smoothly w.r.t. the functional composition. Indeed the realized set-theoretic functions are not even monotonic w.r.t. the order on the quotient space. We wonder if it is possible to get around these drawbacks by adding further constraints to the per's structure and/or to the underlying $\lambda$-model.

6.2. Unsolvable Types

In Amadio (1988) we study a fixed-point extension of the second-order $\lambda$-calculus (Section 2) and we introduce a notion of unsolvable type, i.e., a type whose terms are all observably equivalent to $\perp$, the everywhere undefined term. Model theoretically this condition can be rephrased in the category $c(u)per$ by considering types whose interpretation is a terminal object.

Of course if the model is adequate (i.e., observationally equivalent terms are equated in the model) we have: $[x] \cong 1$ (terminal object) $\Rightarrow x$ unsolvable. Terms having unsolvable types carry no information; hence their value can be statically determined without performing any reduction.
The following proposition determines a rather interesting class of unsolvable types (we were unable to find a syntactic proof of this fact).

Let $\alpha$ be a closed nonrecursive type. We can always reduce $\alpha$ to a semantically equal type of the form $\alpha' = \forall t_1 \cdots \forall t_n \forall t \cdot \sigma_1 \rightarrow \cdots \rightarrow \sigma_n \rightarrow t$, where $\sigma_i(t) \equiv \rho_1 \cdots \rho_{n_i} t$. $\rho_j$ can be either a quantification or of the shape $t \rightarrow$ (every type expression is intended associated to the right).

Let $1_\bot = \{ (\bot, \bot) \}$, it is a terminal object in $\mathfrak{c}(u)_{\mathit{per}}$ and the least $\mathfrak{c}(u)_{\mathit{per}}$ w.r.t. set containment.

6.2.1. PROPOSITION. If for $i = 1, \ldots, n$ and $j = 1, \ldots, n_i$, $\text{Neg}(t, \rho_j)$ (Definition 3.4) then $[\alpha]^{\mathfrak{c}(u)_{\mathit{per}}}$ is a terminal object.

Proof: Observe that the indexed product (or the intersection) of terminal objects is still a terminal object, so it will be enough to prove that the interpretation of: $\beta \equiv \forall t \cdot \sigma_1(t) \rightarrow \cdots \rightarrow \sigma_n(t) \rightarrow t$ is a terminal object; i.e., $\forall t, \forall f, f[\beta] \eta f \Rightarrow f[\beta] \eta \bot$.

By definitions, $f[\beta] \eta f$ iff $\forall i \forall d_i e_i, d_i[\sigma_i]_{\eta(1^i)} e_i = fd_1 \cdots d_n A e_i \cdots e_n$. Consider the $\mathfrak{c}(u)_{\mathit{per}} 1_\bot$ and observe that $A \supseteq 1_\bot \Rightarrow \forall i (d_i[\sigma_i]_{\eta(1^i)} e_i = d_i[\sigma_i]_{\eta(1^i)} e_i)$ (by the hypothesis that $t$ occurs negatively) $\Rightarrow fd_1 \cdots d_n = fe_1 \cdots e_n = \bot$. Q.E.D.

Note. Example: $\forall t \cdot ((\forall s \cdot t \rightarrow s) \rightarrow t)$ is unsolvable. Observe that there is no relation between solvability and provability (i.e., existence of a pure closed term of the given type), e.g.: $\forall t \cdot (t \rightarrow t) \rightarrow t$ is unprovable but solvable (a natural choice for an observable value is: $\lambda t \cdot \lambda x^i \cdot x^i$). By syntactic means it is proved in Amadio (1988) that with reference to 6.2.1 if $\exists i, \sigma_i \equiv \forall t \tau_1 \cdots \tau_n t$ then $\alpha$ is solvable.

6.3. Type Containments

Every recursive type can be "unfolded" so to generate an infinite regular tree according to the following (partially) formal definition. We recall that regular trees are characterized as those trees that have a finite number of subtrees; they have a theory very much related to that of regular languages (see Courcelle, 1983).

6.3.1. DEFINITION. Denote with Tree($\Sigma$) the collection of possibly infinite, finitely branching labelled trees over the alphabet of ranked symbols (the rank being shown as a superscript): $\Sigma = \{ \bot^0, T^0, i^0, s^0, \ldots, \forall t^1, \forall s^1, \ldots, \rightarrow^2/l, s$ type variables}. We denote with $A, B, \ldots$ trees in Tree($\Sigma$).

The following schemas are supposed to suggest a formal definition of the regular tree associated to a recursive type that goes by induction on the structure of the type and the length of the path $\pi \in \omega^*$ whose associated label (if any) we are defining:
RECURSION OVER REALIZABILITY STRUCTURES

\[
T(t) = t \quad T(\alpha \rightarrow \beta) = T(\alpha) \rightarrow T(\beta) \quad T(\forall t \cdot \alpha) = \forall t
\]

\[
T(\mu t \cdot \alpha) = \bot \quad \text{if } \alpha \in \text{Bot}(t), \text{ where } \alpha \in \text{Bot}(t)
\]

\[
T(\mu t \cdot \alpha) = T([\mu t \cdot \alpha/t] \alpha), \quad \text{otherwise, i.e.,}
\]

\[
either t \not\in FV(\alpha) \text{ or } \alpha \equiv Q_t \cdots Q_{t_n} \cdot t, \ Q ::= \forall \mu, n \geq 0.
\]

An ordering on trees. We define \( \leq \) as the least binary relation on \( \text{Tree}(\Sigma) \) satisfying:

\[
A = A \Rightarrow A \leq A; \quad \bot \leq A; \quad A \leq T.
\]

\[
A' \leq A, B \leq B' \Rightarrow A' \leq A \quad \text{and } \forall t \quad \forall t
\]

\[
A \leq B \quad \text{and } B' \leq B \quad \Rightarrow \quad B' \leq B
\]

Let \( A \in \text{Tree}(\Sigma) \), \( \pi \) a path then the tree \( A|_k \) resulting from cutting \( A \) at the \( k \)th level is defined as

\[
A|_k(\pi) = \begin{cases} \text{case}(A(\pi) \uparrow \text{ or } |\pi| < k); & A(\pi); \\ \text{if } |\pi| = k, A(\pi) \downarrow, \text{Pos}(A, \pi)); & \bot; \\ \text{if } |\pi| = k, A(\pi) \downarrow, \text{Neg}(A, \pi)); & T. \end{cases}
\]

where \( |\pi| \) is the path length; \( \text{Pos}(A, \pi) \) (\( \text{Neg}(A, \pi) \)) iff in the path \( \pi \) we select the left subtree of a tree whose root is \( \rightarrow \) an even (odd) number of times (i.e., just the usual notion of positive (negative) occurrence). We say \( A < B \) iff \( \forall k = (A|_k \ll B|_k) \).

6.3.2. LEMMA. (Substitution and unfolding). Let \( n \in \omega, \alpha \) and \( \beta \) types, \( \eta \) type environment. Then:

(1) \( R(n, [\beta/t] \alpha, \eta) = R(n, \alpha, \eta[R(n, \beta, \eta)/t]). \)

(2) \( R(n, \mu t \cdot \alpha, \eta) = R(n, [\mu t \cdot \alpha/t] \alpha, \eta). \)

Proof. (1) By induction on \( n \) and the structure of \( \alpha. \)

(2) Easy corollary of (1). Q.E.D.

6.3.3. THEOREM. Given \( \alpha, \beta \) types if \( T\alpha \ll T\beta \) then \( \forall \eta[\alpha]_n \ll [\beta]_n. \)

Proof. For each type \( \alpha \) let \( (x)_\alpha \) denote a new type whose intended interpretation is: \( [(x)_\alpha]_n = [x]_n \cap D_n^2. \) It has been shown in 5.5 that \( [x]_n \) is the completion of \( (\bigcup_{n<\omega} [(x)]_n). \)

Now we define an interpretation for trees.

To every tree \( A|_k \) there corresponds a unique type and its interpretation
ROBERTO M. AMADIO

(suppose \([\bot]_\eta = \eta_{\{\bot, \bot}\}} \) and \([T]_\eta = \eta_{D^2}\). Let \([A]_\eta\) be the completion of \((U_{k<\omega} [A_{k}]_\eta)\). Of course we have defined the cut in such a way that the interpretation of \(A_{k}\) approximates the interpretation of \(A\).

Next we show \([x]_\eta = [Tx]_\eta\). Observe \(\forall k, [A_{k}]_\eta \subseteq [A_{k+1}]_\eta \subseteq [A]_\eta\). The second inclusion is trivial; for the first one use the compositionality of the semantics and the properties of positive and negative occurrences.

It is not difficult to prove the following fact: \(\forall n \exists N \forall k \geq N, ([x]_\eta = ([Tx]_\eta)\). Therefore: \([x]_\eta \subseteq [Tx]_\eta\). Vice versa, \(\forall k, [Tx_{k}]_\eta \subseteq [x]_\eta\). This is based on: super \(\vdash \mu \cdot \alpha = \bot\) if \(\alpha \in \text{Bot}(t)\), super \(\vdash \mu \cdot \alpha = [\mu \cdot \alpha/t] \alpha\) and the properties of positive and negative occurrences.

Eventually observe: \(Tx < T\beta \iff \forall k(Tx|_k \leq T\beta|_k)\) and \(Tx|_k \leq T\beta|_k \Rightarrow [Tx|_k]_\eta \subseteq [T\beta|_k]_\eta\); hence, \(Tx < T\beta \Rightarrow \forall n[x]_\eta \subseteq [\beta]_\eta\). Q.E.D.

Remarks. (1) (On characterizing valid type-containments) In Amadio (1989) it is shown that as long as we do not consider second-order types and we give an F-semantics to type expressions (see note 1.4) the condition in Theorem 6.3.3 is also necessary. As for pure second-order types Mitchell (1988) presents a characterization of type containments valid in all models. We wonder if it is possible to get a completeness theorem for the combination of the two theories.

(2) Bruce and Longo (1988) propose to interpret the standard relation of subtyping arising in typed, functional languages with multiple inheritance as type containment in per models. Briefly the semantic justification is that: \([x] \subseteq [\beta] \Rightarrow \exists! c: [x] \rightarrow [\beta]\) (identity realizes \(c\)). In words, \(A\) is a subtype of \(B\) iff there is a (unique) morphism from \(A\) to \(B\) (playing the role of a coercion) that is realized by the identity. This is not the right place to start an extended discussion on the semantics of inheritance (see, e.g., Amadio, 1989); however, it should be clear that Theorem 6.3.3 suggests a rule for the subtyping of recursive types; indeed such rule is stronger than other rules in the literature and seems suitable to an efficient mechanization.

CONCLUSIONS

In this paper we try to substantiate the thesis that realizability interpretations provide an interesting basis for the semantics of programming languages, integrating the domain theoretic approach. In particular, we can construct quite interesting and manageable models of typed functional languages.

The models studied give deep insights into the properties of such languages although they suffer from a typical drawback of denotational semantics, namely their equational theories are hard to characterize and typically they are not even r.e.. Therefore there are obvious difficulties
to extract from the models and justify a *finitary programming logic* (see Section 6).

In a sense this seems unavoidable as the properties we are interested in (e.g., program equivalence) are *inherently non-r.e.* Nevertheless, a theory can justify its existence by being successful in deriving a large amount of interesting facts (like Peano arithmetic). An example of this phenomena in the programming field is Scott’s LCF whose design principles are based on domain theory.

It should be recognized though that in the model theory of programming languages there is no such structure that can play the privileged role of, e.g., the natural numbers in arithmetic; hence it is important to study and compare classes of models and derive *completeness theorems.* It is our hope that starting from the concrete models presented here a fruitful process of abstraction of their relevant properties can be carried out.

Going back to our work a variety of technical issues remain to be explored; among other we select the remarks in 1.5.2 on internal definability, 6.1.4 on ordering the quotient space, and 6.3 on type-containments. More generally we think it worthwhile to develop an abstract framework (getting inspiration from previous work on Scott’s domains) that connects such notions as computability, partiality, topology, and ordering.

**Acknowledgments**

My advisor Giuseppe Longo led me to the study of realizability structures. Continuous discussions with him and other attendants of our “internal seminar” in Pisa provided the cultural humus for the growth of this work. As for the interpretation of recursive types I wish to thank Professor Mario Coppo and his student Felice Cardone for a few insightful discussions on the subject. The kind remarks of two anonymous referees have been taken into account in the final revision of the paper.

Received January 9, 1989; final manuscript received October 27, 1989

**References**

AMADIO, R. (1988), A fixed point extension of the second order lambda calculus: Observable equivalences and models, in “3rd IEEE LICS, Edinburgh.”


Cartwright, R. (1984), Types as intervals, in "Proceedings, ACM-POPL '84."


Coppo, M., and Zacchi, M. (1986), Type inference and logical relations, in "1st IEEE LICS, Boston."


Hyland, M. (1982), The effective topos, in "Brouwer Symposium" (Troelstra and Van Dalen, Eds.), North-Holland, Amsterdam.


MOGGI, E. (1986), Interpretation of second order lambda-calculus in categories, manuscript.


