Type Inference with Recursive Types: Syntax and Semantics*

FELICE CARDONE AND MARIO COPPO

Dipartimento di Informatica, Università di Torino,
C.so Svizzera 185, 10149 Torino, Italy

In this paper we study type inference systems for $\lambda$-calculus with a recursion operator over types. The main syntactical properties, notably the existence of principal type schemes, are proved to hold when recursive types are viewed as finite notations for infinite (regular) type expressions representing their infinite unfoldings. Exploiting the approximation structure of a model for the untyped language of terms, types are interpreted as limits of sequences of their approximations. We show that the interpretation is essentially unique and that two types have equal interpretation if and only if their infinite unfoldings are identical. Finally, a completeness theorem is proved to hold w.r.t. the specific model we consider for a natural (infinitary) extension of the type inference system. © 1991 Academic Press, Inc.

1. INTRODUCTION

One of the most interesting notions of type constraint for functional programming languages is the one derived from Curry's Functionality Theory (Curry and Feys, 1958), which has suggested the type disciplines incorporated in some programming languages of recent design, notably ML (Gordon, Milner, and Wadsworth, 1979) and HOPE (Burstall, MacQueen, and Sannella, 1981). In this approach types are assigned to terms of the $\lambda$-calculus according to a set of formal rules which can be effectively checked at compile time. Types describe then the functional behaviour of terms in such a way that, in general, the same term can be assigned infinitely many types depending on the particular program context in which it occurs. For example, the term $\lambda x.x$ can have type $\textbf{bool} \rightarrow \textbf{bool}$, $\textbf{int} \rightarrow \textbf{int}$, $(\textbf{int} \rightarrow \textbf{int}) \rightarrow (\textbf{int} \rightarrow \textbf{int})$, etc. (where $\sigma \rightarrow \tau$ denotes the type of functions mapping values of type $\sigma$ to values of type $\tau$), according to whether it is intended to compute the identity function over boolean values, over integers, or over functions of type $\textbf{int} \rightarrow \textbf{int}$.

This causes a natural notion of implicit polymorphism to be introduced,

* Research supported by EEC Joint Collaboration Contract ST2-0374-C (EDB) and 60% M.P.I.

0890-5401/91 $3.00
Copyright © 1991 by Academic Press, Inc
All rights of reproduction in any form reserved.

48
where type schemes are assigned to terms and they may be instantiated to match the types required by the surrounding program context. Several features make this sort of polymorphism particularly attractive from both the practical and the theoretical point of view. The type inference algorithm is complete, due to the existence of principal type schemes (Hindley, 1969; Milner, 1978) which fully characterize the set of types assignable to each term. Moreover natural notions of interpretation of types in models for the base language can be introduced for which the formal assignment rules are sound and complete, yielding a semantical proof that terms having type in this discipline cannot produce run time errors.

In the present paper we extend this approach to include recursive type definitions as a means of introducing infinite type expressions. The usefulness of this construct was pointed out for the first time (at least to the authors' knowledge) in (Morris, 1968).

For example, the availability of recursively defined types enables one to assign type $(\sigma \rightarrow \sigma) \rightarrow \sigma$, for every type $\sigma$, to Curry's fixed point combinator $Y \equiv \lambda f. (\lambda x. f(xx))(\lambda x. f(xx))$, permitting in this kind of system recursion over values without requiring its explicit introduction in the base language by means of a new constant.

In Section 2 we describe the main syntactical properties of type assignment systems with recursive types: we find it helpful to consider recursive types as finite notations for infinite types of a special kind, namely those whose construction tree is regular in the sense of Courcelle (1983). This allows a smoother presentation of their properties and suggests naturally an equivalence relation between recursive types, defined as the equality of their infinite unfoldings. This relation is stronger than that often adopted in the literature (see, e.g. (MacQueen, Plotkin, and Sethi, 1986)), but is the one which is usually implemented for type checking circular types (see Aho, Sethi, and Ullman, 1986). We discuss the two equivalence relations, mainly for the purpose of observing that the principal type scheme property holds for one of them only, yielding a complete type inference algorithm as an application of the unification algorithm for regular types (Huet, 1976), and thus answering a question asked in (Reynolds, 1985).

Section 3 introduces topological models for the base language on which a fairly general notion of approximation can be defined, making them similar to models constructed as inverse limits (Scott, 1972). Types are interpreted as special subsets of such models, namely ideals, i.e., non-empty closed sets w.r.t. the Scott topology defined on the models. Using this notion, the interpretation of a type can be built as the limit of the denumerable sequence of its approximate interpretations, as in (Coppo, 1985). As a result, we are able by this technique to give interpretation to recursive types without any restriction of the kind introduced in (MacQueen, Plotkin, and Sethi, 1986), necessitated by the particular
technique used in that paper to interpret types. We show the uniqueness of the interpretation of types and prove that our notion of type equivalence completely describes semantical equality of interpretations.

Finally, in Section 4, we prove a completeness property for an extension of the system motivated by the topological nature of our model, namely, that any typing statement true in the model is also provable in the extended system, modifying a technique used in (Coppo, 1984).

2. Types and Type Assignment

In this section, for the purpose of fixing terminology and notational conventions, we give the basic formal definitions concerning recursive (and infinite) types and introduce systems for assigning them to $\lambda$-terms, with an outline of the basic properties needed in the sequel.

We will consider recursive types as a notation to denote infinite types (see, e.g., Morris, 1968). This point of view yields a natural way of dealing with infinite types in a type assignment system, in such a way that many desirable properties usually possessed by this kind of systems (such as the existence of principal type schemes) are preserved.

Syntax of Types

Two common methods for introducing the notion of recursive type are the use of recursive equations or the use of a recursion operator over types: we follow the latter approach for its convenience, in that it will not force us to go outside the system itself to define the types involved in a deduction (see Remark 2.4(i)).

2.1. Definition. Let $K$ be a set of type constants (ranged over by $\kappa$) and $V_T$ a set of type variables (ranged over by $s, t, \ldots$). The set $T_\mu$ of type schemes (which we shall call simply types for short) is the smallest set such that:

1. $K \subseteq T_\mu$
2. $V_T \subseteq T_\mu$
3. If $\sigma, \tau \in T_\mu$, then $\sigma \rightarrow \tau \in T_\mu$
4. If $t \in V_T$, $\sigma \in T_\mu$, then $\mu t. \sigma \in T_\mu$.

As usual, the $\rightarrow$ type constructor associates to the right, so, e.g., $\sigma \rightarrow \rho \rightarrow \tau$ is the same as $\sigma \rightarrow (\rho \rightarrow \tau)$. $\sigma, \tau, \rho$ will denote types in $T_\mu$. All occurrences of the type variable $t$ in $\sigma$ become bound in $\mu t. \sigma$, and a type is closed if it does not contain free occurrences of type variables.

The set $T_F$ of finite types is the subset of $T_\mu$ containing all types without
occurrences of the \(\mu\) operator. A type is \textit{ground} if it belongs to the subset of \(T_F\) generated by type constants, so any ground type is also closed but the converse is not true as infinite closed types are not ground.

A \textit{substitution} is a function \(s: V \rightarrow T_\mu\). The result of substituting \(\tau\) for \(t\) in \(\sigma\) is denoted by \(\sigma[\tau/t]\), where it is assumed that no variable occurring free in \(\tau\) becomes bound in \(\sigma[\tau/t]\) (this can always be achieved by considering types modulo a renaming of bound variables). A substitution \(s\) can be extended to a function in \(T_\mu \rightarrow T_\mu\) by defining \(s(\tau)\) as the type obtained from \(\tau\) by replacing all free occurrences of any type variable \(t\) in \(\tau\) by \(s(t)\). A substitution \(s\) is \textit{closed} (resp. \textit{ground}) if \(s(t)\) is a closed type (resp. ground type) for all type variables \(t\).

It is a standard practice to consider expressions, in particular type expressions, as linear descriptions of their construction trees. This convention has the advantage, from the point of view of the present paper, of being smoothly extendible to expressions whose construction trees have infinite height.

In the sequel we shall denote by \(T_\infty\) the set of finite and infinite labelled trees with labels over \(K \cup \{\Omega\} \cup V_T \cup \{\rightarrow\}\), where \(\Omega\) is a new type constant standing for the "undefined" tree. Each label is equipped with a natural number, its \textit{arity}, which is 0 for type constants, \(\Omega\) and type variables, and 2 for \(\rightarrow\), and we assume that every node in a tree has a number of immediate descendants given by the arity of its label. We refer the reader to (Courcelle, 1983) for a detailed description of properties of infinite trees.

An \textit{infinite type} will be an element of \(T_\infty\). As an example, consider

\[
\chi \equiv \rightarrow t
\]

\[
\rightarrow t
\]

\[
\vdots
\]

Note that \(\chi\) can be considered as the infinite unfolding of the recursive type \(\mu s.s \rightarrow t\) (which consists in performing infinitely many times the substitution of \(\mu s.s \rightarrow t\) for \(s\) in \(s \rightarrow t\)), or equivalently as the solution of the equation \(\chi = \chi \rightarrow t\).

A \textit{subtype} of a (possibly infinite) type \(\alpha \in T_\infty\) is the element of \(T_\infty\) which consists of a node of \(\alpha\) together with all its descendants (in \(\alpha\)). A (possibly infinite) type is \textit{regular} if it has a finite set of subtypes; as an obvious consequence we have that every type in \(T_F\) is regular, and an example of infinite regular type is given by \(\chi\) defined before (notice that \(\chi\) has only two distinct
subtypes, the type variable \( t \) and itself). \( T_R \) will denote the set of regular types. It is clear how to extend the definitions of substitution and ground substitution to regular types. We point out that regular types are closed under regular substitutions, where a substitution is regular if its range is a subset of \( T_R \).

It is known (see Courcelle, 1983, Sect. 2.3) that \( T_\infty \) can be partially ordered by a relation \( \leq \) which satisfies the conditions

- \( \Omega \leq \alpha \) for any type \( \alpha \),
- \( \alpha \rightarrow \beta \leq \alpha' \rightarrow \beta' \) iff \( \alpha \leq \alpha' \) and \( \beta \leq \beta' \),

and with respect to this partial order \( T_\infty \) has a least element \( \Omega \) and least upper bounds of countable directed subsets. Also, it can be shown that the mapping

\[
F \equiv (\lambda \xi \in T_\infty. \alpha[\xi/t]) : T_\infty \rightarrow T_\infty
\]

is continuous (i.e., it is monotonic and preserves least upper bounds of countable directed subsets of \( T_\infty \)), so it has a least fixed point \( \text{fix}(F) \), defined as the least upper bound of the \( \leq \)-increasing sequence \( \langle F^{(n)}(\Omega) \rangle_{n \in \omega} \) (where \( F^{(n)} \) is the \( n \)-fold composition of \( F \) with itself). Furthermore, \( \text{fix}(F) \) is regular if \( \alpha \) is a regular tree (see Courcelle, 1983, Theorem 4.10.1).

Recursive types will be viewed in this paper as a linear notation to denote regular types. This interpretation is justified by the property that for each recursive type there is a unique regular type associated with it, according to the following translation:

2.2. Definition. (i) \((-)\ast : T_\mu \rightarrow T_R\) is defined inductively by

1. \( t\ast = t \) (\( t \in V_\tau \))
2. \( \kappa\ast = \kappa \) (\( \kappa \in K \))
3. \( (\sigma \rightarrow \tau)\ast = \sigma\ast \rightarrow \tau\ast \)
4. \( (\mu t. \sigma)\ast = \text{fix}(\lambda \xi \in T_\tau. \sigma\ast[\xi/t]) \).

(ii) \( \approx \leq T_\mu \times T_\mu \) is the equivalence relation defined by \( \sigma \approx \tau \) iff \( \sigma\ast = \tau\ast \).

From the remarks before it follows that the translation is well defined for all types in \( T_\mu \). Conversely, Braquelaire and Courcelle (1984, Corollary 5.6) show that every regular type has a notation in \( T_\mu \), and this fact allows us, when convenient, to prove facts about \( T_\mu \) using properties of \( T_R \) (see, e.g., Theorem 2.9). \((-)\ast \) is a surjective mapping but is not one-one. because for every regular type \( \alpha \in T_\mu \) there are in general many \( \sigma \in T_\mu \) such that \( \sigma\ast = \alpha \). For example \( \chi = (\mu s. s \rightarrow t)\ast = (\mu s. (s \rightarrow t) \rightarrow t)\ast \) (i.e.,
\[\mu.s \to t \equiv \mu.s.(s \to t) \to t.\] However, as a consequence of (Braquelaire and Courcelle, 1984), we have that for each type \(x \in T_R\) there is a type \(\sigma \in T_\mu\) with a minimal number of symbols such that \(\sigma^* = x\). For instance \(\mu.s.s \to t\) is the simplest recursive type representing \(x\). It is also important to remark that the relation \(\approx\) is decidable by the simple algorithm described in (Courcelle, Kahn, and Vuillemin, 1974, Theorem 2).

Finally, observe that \(\sigma^* = \Omega\) iff \(\sigma = \mu t_1 \cdots \mu t_n.t\), for some \(n \in \omega\) and \(1 \leq i \leq n\). Types of this form were excluded in (MacQueen, Plotkin, and Sethi, 1986) on the basis of semantical considerations.

**Type Assignment Rules**

We turn now to the description of the type assignment system. The set \(A\) of type-free terms (ranged over by \(M, N, \ldots\)) is defined from a set \(I'\) of formal term variables (ranged over by \(x, y, \ldots\)) and a (possibly empty) set \(C\) of term constants (ranged over by \(c\)) in the following way:

\[M ::= x | c | MN | \lambda x.M.\]

\(FV(M)\) denotes the set of variables free in \(M\). A pure \(\lambda\)-term is a term without occurrences of constants (see Barendregt, 1984 for further explanations about \(A\) and the notational conventions followed in the present paper).

A statement is an expression \(M:\tau (\tau \in T_\mu)\), whose meaning is that the type \(\tau\) is assigned to the \(\lambda\)-term \(M\). A basis \(B\) is a finite set of statements of the form \(x:\sigma\) such that no (term) variable occurs more than once in \(B\). \(B[x:\tau]\) represents the basis \(B \setminus \{x:\sigma\} \cup \{x:\tau\}\) (where \(\setminus\) denotes set difference and \(\sigma\) is any type). Let \(\tau_c:C \to T_\mu\) be a function which assigns a type to each constant.

In the formulation of type assignment rules we follow the approach of (Damas and Milner, 1982; MacQueen, Plotkin, and Sethi, 1986). This is essentially a natural deduction formulation, in which statements carry with them the set of assumptions on which they depend. More precisely, following (MacQueen, Plotkin, and Sethi, 1986), let a sequent be an object of the shape \(B \vdash_x M:\tau\).

2.3. Definition. The type assignment system \(\vdash_\mu\) is defined by the following rules:

\[
\begin{align*}
B[x:\sigma] & \vdash_\mu x:\sigma & \text{(Var)}
\end{align*}
\]

\[
\begin{align*}
B & \vdash_\mu c: \tau_c(c) & \text{(Const)}
\end{align*}
\]
\[
\begin{align*}
B[x:\sigma] & \vdash \mu M: \tau \\
B & \vdash \mu \lambda x. M: \sigma \rightarrow \tau & (\rightarrow I) \\
B & \vdash \mu M: \sigma \rightarrow \tau \\
B & \vdash \mu N: \sigma & (\rightarrow E) \\
B & \vdash \mu (MN): \tau \\
B & \vdash \mu M: \sigma & \sigma \approx \tau \\
B & \vdash \mu M: \tau & (\approx).
\end{align*}
\]

2.4. Definition and Remark (i) For the purpose of motivating these rules it is expedient to introduce an auxiliary system, denoted by \( \vdash_R \), for explicitly assigning infinite regular types to \( \lambda \)-terms. So the types of \( \vdash_R \) are elements of \( T_R \) and its only rules are (Var), (Const), (\( \rightarrow I \)), (\( \rightarrow E \)) where \( \sigma, \tau \) are now taken to range over \( T_R \). In this system, for example, a term of type \( \chi \), defined above, can be applied to itself using only rule (\( \rightarrow E \)) since \( \chi \) is identical to \( \chi \rightarrow t \). The system \( \vdash_\mu \) is then equivalent to \( \vdash_R \) in the sense that

\[
B \vdash_\mu M: \tau \quad \text{if and only if} \quad B^* \vdash_R M: \tau^*,
\]

where, if \( B = \{x_1: \sigma_1, \ldots, x_n: \sigma_n\} \), then \( B^* = \{x_1:(\sigma_1)^*, \ldots, x_n:(\sigma_n)^*\} \).

(ii) If we restrict the set of types to \( T_F \) and we drop rule (\( \approx \)) we obtain the classical Curry type assignment system, which we denote with \( \vdash_F \). However, the system \( \vdash_\mu \) is not conservative over \( \vdash_F \), in the sense that there are finite types which turn out to be assignable to a \( \lambda \)-term in \( \vdash_\mu \) but not in \( \vdash_F \). For example, the deduction

\[
\begin{array}{c}
\phi \vdash_{\mu} x:(\mu.s \rightarrow t)
\end{array}
\]

shows that \( \lambda x.xx \) has type \( (\mu.s \rightarrow t) \rightarrow t \) for any type variable \( t \). Now observe that \( (\mu.s \rightarrow t) \rightarrow t \approx (\mu.s \rightarrow t) \rightarrow t \) so that we have a deduction

\[
\begin{array}{c}
\phi \vdash_{\mu} \lambda x.xx:(\mu.s \rightarrow t)
\end{array}
\]

but no type can be assigned to \( (\lambda x.xx)(\lambda x.xx) \) in Curry's type assignment system \( \vdash_F \).

Main Properties

We now present some basic syntactical properties of the systems \( \vdash_\mu \). It will turn out that most results holding for Curry's type assignment system are still valid in \( \vdash_\mu \).
However, one main feature in which $\beta_\mu$ differ from $\beta_F$ is the fact that the normal form theorem fails for it. We have seen, e.g., that $\emptyset \beta_\mu (\lambda x.x)(\lambda x.x) : \tau$ (recall that $(\lambda x.x)(\lambda x.x)$ has no normal form).

(A characterization of the cases in which strong normalization holds even when recursive types are allowed is given in (Mendler, 1987), using the weaker notion of equivalence $\sim$ defined below).

Moreover let $\xi = \mu t.t \to t$. It is easy to check that $B_\xi \beta_\mu M : \xi$ for all $\lambda$-terms $M$ without occurrences of constants, where $B_\xi = \{ x : \xi \mid x \in V \}$ (note that, for any type $\tau$ without occurrences of type constants, there always exists a substitution $s$ such that $s(\tau) \approx \xi$: it is sufficient to take $s(t) = \xi$ for all $t \in V_\tau$).

Let $\longrightarrow_\beta$, $\longrightarrow_\beta_\eta$ respectively denote the relations of $\beta$- and $\beta_\eta$-reduction (see Barendregt, 1984, Chap. 4). A first property, which can be proved in a standard way, is that types are preserved under $\longrightarrow_\beta_\eta$.

2.5. **Lemma** (Subject Reduction). If $B \beta_\mu M : \tau$ and $M \beta_\eta N$ then $B \beta_\mu N : \tau$.

2.6. **Remark.** (i) The converse of 2.5, as for $\beta_F$, is not true; i.e., if $B \beta_\mu N : \tau$ and $M \beta_\eta N$ we do not have, in general, $B \beta_\mu M : \tau$.

(ii) If it is possible to assign a type to a term $M$, then every subterm of it has a type and, by Lemma 2.5, every term $N$ such that $M \beta_\mu N$ has all types assignable to $M$ (and possibly others): this fact entails that no incorrect applications will be created during a computation starting at $M$, such as the application of a constant, say of type $\text{int} \to \text{int}$, to a term which does not have type $\text{int}$ (where $\text{int}$ represents the basic type of integers). This fact can be seen as a syntactic version of the soundness theorem of (Milner, 1978) (a semantical proof similar to that of (Milner, 1978) is, of course, possible also in our case—see Theorems 3.9, 4.2). So even a type like $\xi$ carries some meaningful information: namely that during the evaluation of a term having this type no error depending on incorrect applications of the kind described above will occur.

A simple induction on derivations shows that type assignment is closed under substitution for type variables:

2.7. **Lemma.** If $B \beta_\mu M : \tau$ then $s(B) \beta_\mu M : s(\tau)$.

A main property of $\beta_\mu$ is the existence of principal type schemes. In the definition of this notion for our system we must take into account the relation $\approx$. If $B$ is a basis, let $B \uparrow M$ be the restriction of $B$ to the free variables of $M$. It is easy to prove that if $B \beta_\mu M : \tau$ and $B \subseteq B'$ then $B' \beta_\mu M : \tau$, and that $B \beta_\mu M : \tau$ implies $B \uparrow M \beta_\mu M : \tau$. If $B_1 = \{ x_1 : \sigma_1, ..., x_n : \sigma_n \}$, $B_2 = \{ x_1 : \tau_1, ..., x_n : \tau_n \}$, then $B_1 \approx B_2$ means that $\sigma_i \approx \tau_i$ for all $1 \leq i \leq n$. 

2.8. **Definition.** (i) A basis $B_p$ and a type $\tau$ are *principal* for a term $M$ iff:

1. $B_p \vdash_M M : \pi$
2. $B_p = B_p \upharpoonright M$
3. for all bases $B$ and types $\tau$ such that $B \vdash_M M : \pi$ there is a substitution $s$ such that $(s(B_p)) \approx B' \subseteq B$ for some basis $B'$ and $\tau \approx s(\pi)$.

2.9. **Theorem** (Principal Type Scheme Property). If $B \vdash_M M : \tau$ for some basis $B$ and type $\tau$ then there exists a principal basis $B_p$ and a principal type $\pi$ for $M$. Moreover $B_p$ and $\pi$ can be found in an effective way.

The theorem follows from the fact that the system $\vdash_R$ described in Remark 2.4(i) has this property. In fact, the key point of the proof of the corresponding result for the finite type assignment system given in (Curry, 1969) (see also Hindley, 1969; Damas, 1985) is the unification algorithm. It is well known (see Huet, 1976; Courcelle, 1983, Proposition 4.9.2) that this algorithm can be extended to unification of regular trees (of which our regular types are a special case), with all other details of the proof holding without modifications, as remarked also in (Wand, 1987). From the equivalence of the systems $\vdash_R$ and $\vdash_\mu$, we have the desired conclusion.

A type checking algorithm can be defined using a suitable representation of regular types (for instance, as cyclic graphs) and a unification algorithm which exploits minimal representations of recursive types (see, for instance, (Aho, Sethi, and Ullman, 1986]). For example, it turns out that $(t \rightarrow t) \rightarrow t$ is the principal type scheme of Curry's fixed point combinator $Y = \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$ and $t$ that of $(\lambda x.xx)(\lambda x.xx)$.

Moreover, since every pure term has a type, every pure term has a principal type and basis scheme. (Hindley, 1969) has shown that, in the finite case, every type which is a type scheme of a closed term is also the principal type scheme of some closed term reducible to it. This yields a similar property in the present context also, where we have that every finite type scheme is the principal type scheme of some closed $\lambda$-term. Explicitly:

2.10. **Property.** If $\sigma \in T_F$ then there is a closed term $M$ such that $\sigma$ is the principal type scheme of $M$.

**Proof.** By induction on the complexity of $\sigma$. If $\sigma$ is a type variable $t$, then we have seen that it is the principal type scheme of $(\lambda x.xx)(\lambda x.xx)$. Now assume that $\sigma \equiv \sigma_1 \rightarrow \sigma_2$. By induction hypothesis $\sigma_2$ is the principal type scheme of some closed term $Q$, and the type $\sigma_2 \rightarrow (\sigma_1 \rightarrow \sigma_2)$ is a type scheme for the combinator $K \equiv \lambda xy.x$. By Hindley's result mentioned
before we have that $\sigma_2 \rightarrow (\sigma_1 \rightarrow \sigma_2)$ is the principal type scheme of a closed term $P$ (reducible to $K$), so $\sigma_1 \rightarrow \sigma_2$ is principal for $M = PQ$. 

We conjecture that the property holds for all types $\sigma \in T_R$.

2.11. Remarks. (i) Sometimes, especially on the basis of semantical considerations, it is necessary to have the same types assignable to terms which are in the $\beta$- or $\beta\eta$-convertibility relation. Consider for instance the terms

$$SII: \mu t. t \rightarrow t \quad \text{and} \quad I: s \rightarrow s,$$

where $S = \lambda xyz. xz(yz)$ and $I = \lambda x.x$. They are $\beta$-convertible (and thus equal in every model) but the indicated type schemes are the principal ones for them, so they have different sets of assignable types. To remedy this we can introduce a system $\vdash_{\mu, Eq}$ obtained from $\vdash_{\mu}$ by adding the rule

$$\begin{align*}
\frac{B \vdash_{\mu, Eq} M : \alpha \quad M = \beta N}{B \vdash_{\mu, Eq} N : \alpha} \quad \text{(Eq)}.
\end{align*}$$

A system $\vdash_{R, Eq}$ can be obtained in a similar way, by adding rule (Eq) to $\vdash_{R}$.

(ii) It is well known that in the finite system $\vdash_{F, Eq}$ the principal type scheme property holds, as in $\vdash_{F}$. The proof of this uses in an essential way the fact that every term having a type in this system has a normal form.

In $\vdash_{R, Eq}$, as well as in $\vdash_{\mu, Eq}$, in contrast, this property is lost. A counterexample can be given by considering the sequence of regular types defined as

$$\alpha_0 = (\mu s. t_0 \rightarrow s)^*,$$

$$\alpha_{n+1} = t_{n+1} \rightarrow \alpha_n,$$

where $t_1, \ldots, t_n, \ldots$ are distinct type variables.

Observe that $\alpha_n = S(\alpha_{n+1})$, where $S$ is the substitution such that

$$S(t_i) = \text{if } i = 0 \text{ then } t_0 \text{ else } t_{i-1}$$

and that $S$ is not invertible (that is, there is no substitution $S'$ such that $\alpha_{n+1} = S'(\alpha_n)$).

Now let $K = \lambda x. \lambda y. x$. We have

$$\begin{align*}
\mathbf{YK} \xrightarrow{\beta} (\lambda x. K(xx))(\lambda x. K(xx)) \xrightarrow{*} \lambda x_1 \cdots x_n . (\lambda x. K(xx))(\lambda x. K(xx))
\end{align*}$$

for every $n \geq 0$ and the principal type scheme of $\lambda x_1 \cdots x_n . (\lambda x. K(xx))$ ($\lambda x. K(xx)$), in $\vdash_{R}$, is $\alpha_{n+1}$. Moreover $\vdash_{R, Eq} \mathbf{YK} : \alpha_n$ for all $n \geq 0$. 
Now assume that $\alpha$ is the p.t.s. of $YK$ in $\vdash_{R, \text{Eq}}$. Then each $\alpha_i$ $(i \geq 0)$ must be an instance of $\alpha$, so $\alpha$ should have the form (modulo a relabeling of type variables):

$$
\begin{array}{c}
\alpha \equiv \\
\begin{array}{c}
\vdash t_0 \\
\vdash t_1 \\
\vdash t_2 \\
\vdash \vdots \\
\end{array}
\end{array}
$$

which is not a regular type. $\alpha$ could be seen, in a limiting sense, as the p.t.s. of $YK$ in a system with rule (Eq) extended to allow non-regular types.

Note that $\pi.t.u \preceq \pi.t.u[t]$, but $\preceq$ is not the minimal congruence with respect to type constructors having this property: for instance we have seen that $\mu.s \rightarrow t \approx \mu.s.(s \rightarrow t) \rightarrow t$, but this fact cannot be proved by substitution alone. Indeed, a weaker notion of equivalence on recursive types can be introduced.

2.12. Definition. Let $\sim \subseteq T_\mu \times T_\mu$ be the smallest equivalence relation satisfying

1. $\mu.\sigma \sim \sigma[\mu.\sigma/t]$  
2. $\sigma \sim \sigma'$ and $\tau \sim \tau' \Rightarrow \sigma \rightarrow \tau \sim \sigma' \rightarrow \tau'$  
3. $\sigma \sim \sigma' \Rightarrow \mu.\sigma \sim \mu.\sigma'$.

For instance, types of the form $\mu.t_1 \cdots \mu.t_n, t_i$ are all $\sim$ equivalent to $\mu.t$. In general $\sigma \sim \tau$ implies $\sigma \approx \tau$ but the converse does not hold. For this reason we shall sometimes refer to these two notions of equivalence respectively as the weak and the strong one.

2.13. Remark. Let $\vdash_{\mu, \sim}$ be the system obtained by replacing $\approx$ with $\sim$ in $\vdash_{\mu}$. $\vdash_{\mu, \sim}$ is a slight modification of the type assignment system defined by (MacQueen, Plotkin, and Sethi, 1986). This system, however, lacks some properties of the system $\vdash_{\mu}$, in particular the principal type scheme property. For instance, let $X = \lambda x.x(\lambda y.yx)$. The principal type scheme of $X$ in $\vdash_\mu$ is $\mu.s.s \rightarrow t$. Now we have $\vdash_{\mu, \sim} X; \mu.s.s \rightarrow t$ and $\vdash_{\mu, \sim} X; \mu.s.((s \rightarrow t) \rightarrow t) \rightarrow t$, where $\mu.s.((s \rightarrow t) \rightarrow t) \rightarrow t$ and $\mu.s.s \rightarrow t$ are not equivalent under $\sim$ and neither is an instance of the order (but note that $\mu.s.((s \rightarrow t) \rightarrow t) \rightarrow t \approx \mu.s.s \rightarrow t$).
Theorem 2.9 and Remark 2.13 answer a question asked by J. C. Reynolds in (Reynolds, 1985) about the existence of principal type schemes in systems of type assignment with recursive types.

Finally we mention a technical result that will be useful in the proof of the completeness theorem in Section 4. The easy proof is left to the reader.

2.14. LEMMA. (i) Let $\sigma, \tau \in T, \sigma \approx \tau$ iff for all ground substitutions $s$, $s(\sigma) \approx s(\tau)$.

(ii) If, for all ground substitutions $s$, $s(B) \rightarrow^\mu M; s(\sigma)$ then $B \rightarrow^\mu M; \sigma$.

3. SEMANTICS OF TYPES

In this section we describe the interpretation of recursive types in topological models of our untyped base language, i.e., models whose underlying set $D$ is a domain satisfying an isomorphism of the kind $$[*] D \cong At + W + [D \rightarrow D]$$

where $+$ stands either for disjoint or coalesced sum of domains, $At$ is (a sum of) a finite collection of domains of basic values like the flat domain of integers $\mathbb{N}^+$ or the flat domain of boolean values $\mathbb{B}^+$, $W = \{?\}$ is the one-element domain used to model run time errors, and $[D \rightarrow D]$ is the space of all continuous functions from $D$ to $D$.

An interpretation of types as special subsets of $D$ was first proposed by Milner in (Milner, 1978), where only finite non-recursive types are considered. In that paper types are interpreted as ideals over $D$, i.e., nonempty subsets of $D$ which are downward closed w.r.t. the partial ordering of $D$ and closed with respect to least upper bounds of their directed subsets. The problem of extending this interpretation to recursively defined types has been solved, using different techniques, by (MacQueen, Plotkin, and Sethi, 1986; Coppo, 1985). In (MacQueen, Plotkin, and Sethi, 1986) the interpretation of a recursive type is found as the unique fixed point of a contractive map on the metric space of ideals (endowed with a suitable metric), while in (Coppo, 1985) (as in the present paper) we define it as the limit of a denumerable sequence of approximate interpretations, built following the approximation structure of $D$.

A natural consequence of our construction is that in the present context interpretations of types are in some sense forced to be ideals over $D$. The closure properties of ideals can be reflected smoothly on the syntactical level suggesting directly an extension of the type assignment system for which a completeness theorem will be proved to hold in the next section.
Furthermore, we show that the equivalence relation $\approx$ completely describes type equality in continuous models.

Finally, our construction of type interpretation can easily be extended to other type constructors like the ones considered by MacQueen, Plotkin, and Sethi, i.e., $+$ (sum of types), $\times$ (cartesian product), $\wedge$ (intersection), $\lor$ (union), $\forall$ (universal quantification over type variables), and $\exists$ (existential quantification over type variables).

**Complete Partial Orders with Approximate Application**

Some acquaintance with domain theory and with the inverse limit construction is assumed (see for instance Scott, 1972, and Scott, 1982). A useful survey of domain construction is also given in (MacQueen, Plotkin, and Sethi, 1986).

A complete partial order (c.p.o.) is a partially ordered set $D$ with a least element $\bot$, and such that every directed subset $X$ has a least upper bound $\bigcup X$. As is well known, the category of c.p.o.’s is closed under a wide range of constructors and, in particular, the space of all continuous functions between two c.p.o.’s is still a c.p.o., where $f:D \to E$ is continuous (with respect to the *Scott topology*) if it is monotonic and preserves least upper bounds of directed sets.

We will interpret our untyped base language in a c.p.o. satisfying an isomorphism of the kind $[*]$. We will assume that $D$ has a notion of approximation with some properties which are satisfied in a c.p.o. obtained as an inverse limit, like Scott’s original $D_\omega$ construction (Scott, 1972).

To simplify notations we will identify in the sequel elements of the components of $D$ (i.e., $At$, $W$, and $[D \to D]$) with their images in $D$ under the isomorphism. So, for example, we will identify $f \in [D \to D]$ with $\text{in}_D(f) \in D$ (where $\text{in}_D$ is the injection of $[D \to D]$ into $D$).

**3.1. Definition.** Let $D$ be a c.p.o. satisfying $D \cong At + W + [D \to D]$.

(i) A notion of approximation over $D$ is a denumerable family of continuous functions $(-)_n:[D \to D]$ such that, for each $n \in \omega$,

1. $d_0 = \bot$ for all $d \in D$
2. $(d_m)_n = (d_m)_n = d_{\min(m,n)}$ $(n,m \in \omega)$
3. $d = \bigcup \{d_n \mid n \in \omega\}$
4. If $a \in At$ then $(a)_{n+1} = a$.

(ii) $D$ has approximable application if there is a notion of approximation over $D$ such that, for all $f \in [D \to D]$,

1. If $n \leq k$ then $f_{n+1}(d_k) = f_{n+1}(d_n)$
2. If $n \leq k$ then $(f_{k+1}(d_n))_n = f_{n+1}(d_n)$
3. $f_{n+1}(d_n) = f_{n+1}(d) = (f(d_n))_n$. 
As observed before, a c.p.o. with approximable application can be obtained by the classical inverse limit construction devised by (Scott, 1972). In this case the mappings \((-\)) are given by \(i_n, \circ j_{n, m}\) where \(i_n, \) and \(j_{n, m}\) are, respectively, the embedding of \(D_n\) in \(D\) and the projection of \(D\) onto \(D_n\) (see Barendregt, 1984, Lemmas 18.2.8, 18.2.9, and Proposition 18.2.13(i)). There are, however, other constructions which yield c.p.o.'s with approximable application, such as the one based on Scott's Information Systems (Scott, 1982).

For the definition of type interpretation we need not assume that \(D\) is algebraic. This assumption will be needed only later. From now on let \(D\) be a c.p.o. satisfying \([\ast]\) and having approximable application.

If \(A \subseteq D\), \(A_n\) is defined by \(A_n = \{d_n \mid d \in A\}\). It is easy to verify that \(D_n\) is a c.p.o. (w.r.t. the ordering inherited from \(D\)) and that \(n \leq k\) implies \(D_n \subseteq D_k\). We will sometimes use the notation \(d_n\) also to denote a generic element of \(D_n\).

An ideal over \(D\) is a subset \(I \subseteq D\) such that

1. if \(e \in I\) and \(d \subseteq e\) then \(d \in I\) and
2. if \(\{d_i\}_{i \in I}\) is a directed subset of \(I\) then \(\bigcup_{i \in I} \{d_i\} \subseteq I\).

Ideals are non-empty closed subsets w.r.t. the Scott topology over \(D\). It is immediate to verify that if \(I\) is an ideal over \(D\), \(I_n\) is an ideal over \(D_n\), but remark that \(I_n\), seen as a subset of \(D\), is not in general an ideal over \(D\), due to the failure of condition (1).

We can now define operations \(\rightarrow\) and \(\Rightarrow\) over subsets of \(D\) and \(D_n\).

3.2. Definition. (i) Let \(A, B\) be two subsets of \(D\). \(A \rightarrow B \subseteq D\) is defined by

\[
A \rightarrow B = \{d \in D \mid d = \perp_D \text{ or } d \in [D \rightarrow D] \text{ and } \forall e \in A \ d(e) \in B\}.
\]

(ii) Let \(A_n, B_n\) be two subsets of \(D_n\). \(A_n \rightarrow^{n+1} B_n \subseteq D_{n+1}\) is defined by

\[
A_n \rightarrow^{n+1} B_n = \{d \in D_{n+1} \mid d = \perp_D \text{ or } d \in [D \rightarrow D] \text{ and } \forall e \in A_n \ d(e) \in B_n\}.
\]

Observe that \(\perp_D\) has been added to cover the case in which \(D\) is taken to be a disjoint sum of domains. This addition is not necessary when we use coalesced sum. Our definition of \(A \rightarrow B\) corresponds to a naive notion of type semantics which is often referred to as the “simple” semantics of types: this is the most widely used in the literature (see, for instance, Milner, 1978; MacQueen, Plotkin, and Sethi, 1986), but we point out that the technique presented in this paper works also with other notions of type semantics such as the “quotient set semantics” in which types are inter-
preted as partial equivalence relations over \( D \) (see Coppo and Zacchi, 1986).

The sets of ideals over \( D \) and \( D_n \) are closed, respectively, under \( \rightarrow \) and \( \wedge \).

3.3. Lemma. (i) If \( A_n, B_n \) are ideals over \( D_n \) then \( A_n \rightarrow B_n \), is an ideal over \( D_{n+1} \).

(ii) If \( A \) and \( B \) are ideals over \( D \) then \( A \rightarrow B \) is an ideal over \( D \).

(iii) Let \( A, B \) be ideals over \( D \). Then \( (A \rightarrow B)_{n+1} = A_n \rightarrow B_n \).

Proof. (i) Let \( d \in A_n \rightarrow B_n \) and \( e \subseteq d \) (where \( e \in D_{n+1} \)). For all \( a \in A_n \), \( e(a) \subseteq d(a) \in B_n \), so \( e(a) \in B_n \) since \( B_n \) is an ideal in \( D_n \). From this it follows that \( e \in A_n \rightarrow B_n \). Now, let \( \{e^{(i)}\}_{i \in I} \) be a directed subset of \( A_n \rightarrow B_n \), and let \( e = \bigcup_{i \in I} e^{(i)} \). If \( a \in A_n \), \( e(a) = (\bigcup_{i \in I} e^{(i)}(a)) = \bigcup_{i \in I} e^{(i)}(a) \) (by continuity of application), \( \{e^{(i)}(a)\}_{i \in I} \) is a directed subset of the ideal \( B_n \), so \( \bigcup_{i \in I} e^{(i)}(a) \in B_n \). Then \( e \in A_n \rightarrow B_n \).

(ii) The proof is by a similar argument.

(iii) \( \subseteq \) If \( d_{n+1} \in (A \rightarrow B)_{n+1} \) then \( d_{n+1} \in A \rightarrow B \), so given

\( a \in A_n \subseteq A \), \( d_{n+1}(a) \in B \). But \( d_{n+1}(a) = (d_{n+1}(a_n))_n \in B_n \) (by Definition 3.1(ii-3)) and we are done.

\( \supseteq \) Let \( d \in A_n \rightarrow B_n \). If \( d = \bot \) the result is immediate. Else, if \( a \in A \), we have \( d(a) = d_{n+1}(a) = d_{n+1}(a_n) \in B_n \subseteq B \) (again by Definition 3.1(ii-3)), so \( d_{n+1} \in (A \rightarrow B)_{n+1} \).

Type Interpretations

We first introduce a general notion of type interpretation in ideal semantics. A type interpretation is parametrized over the ideal \( I \) which interprets all types equivalent to \( \mu t \) (i.e., all types of the shape \( \mu t_1 \cdots \mu t_n \cdot t_i \) for \( 1 \leq i \leq n \)). A natural choice for \( I \) could be the ideal \( \{ \bot \} \) but it will be apparent in the next section that there is no reason to restrict our type interpretations to satisfy such a requirement, so we have chosen to keep this level of generality.

Let \( \{ \kappa_1, \ldots, \kappa_m \} \) be the set of type constants. We assume \( \mathcal{A} = \mathcal{K}_1 + \cdots + \mathcal{K}_m \), where \( \mathcal{K}_i \) is the basic c.p.o. corresponding to \( \kappa_i \) (\( 1 \leq i \leq m \)) in the obvious way. In the sequel \( \mathcal{I}, \mathcal{I}_n \) will denote, respectively, the collections of all ideals over \( D \) and \( D_n \) which do not contain the error element? A type environment \( \eta \) is a function \( \eta: V_T \rightarrow \mathcal{I} \). We use \( T_{\text{env}} \) to denote the collection of type environments.

3.4. Definition. Let \( I \in \mathcal{I} \). An \( I \)-type interpretation (when the ideal \( I \) is understood from the context) is a function \( \mathcal{I}^I[\[ - \]]: T_\mu \rightarrow T_{\text{env}} \rightarrow \mathcal{I} \) such that
1. \( \mathcal{T}'[t] \eta = (\eta(t)) \)
2. \( \mathcal{T}'[\kappa_i] \eta = K_i \cup \{ \perp_D \} \) (1 \( \leq i \leq m \))
3. \( \mathcal{T}'[\mu t] \eta = I \)
4. \( \mathcal{T}'[\sigma \rightarrow \tau] \eta = \mathcal{T}'[\sigma] \eta \rightarrow \mathcal{T}'[\tau] \eta \)
5. \( \sigma \approx \tau \Rightarrow \forall \eta \in T_{env}, \mathcal{T}'[\sigma] \eta = \mathcal{T}'[\tau] \eta. \)

Note that in Definition 3.4 we have only required the interpretation of types to preserve the weaker equivalence \( \sim \) of Definition 2.12. It will turn out that, in topological models with approximable application, \( \sigma \approx \tau \iff \forall \eta \in T_{env} \). We will build a particular type interpretation \( \mathcal{T}'[-] \) via the mapping \((-)^*\) by giving an interpretation of types in \( T_R \).

Definition 3.4 cannot be considered an inductive definition of a type interpretation due to point 5. We will build a particular type interpretation \( \mathcal{T}'[-] \) via the mapping \((-)^*\) by giving an interpretation of types in \( T_R \). As previously remarked this interpretation is defined through its approximations \( \mathcal{T}'[-] \) in \( D_n \).

**3.5. Definition.** (i) Let \( \alpha \in T_R \). The interpretation \( \mathcal{T}'[-]: T_R \rightarrow T_{env} \rightarrow I_1 \) is defined (by induction on \( n \)) by

1. \( \mathcal{T}'[-][\alpha] \eta = \{ \perp_D \} \) for all \( \alpha \in T_R \)
2. \( \mathcal{T}'[-][\kappa_i] \eta = K_i \cup \{ \perp_D \} \) (1 \( \leq i \leq m \))
3. \( \mathcal{T}'[-][\Omega] \eta = I_{n+1} \)
4. \( \mathcal{T}'[-][\mu t] \eta = (\eta(t))_{n+1} \)
5. \( \mathcal{T}'[-][\alpha \rightarrow \beta] \eta = \mathcal{T}'[-][\alpha] \eta \rightarrow \mathcal{T}'[-][\beta] \eta \)

(ii) Let \( \alpha \in T_R \). \( \mathcal{T}'[-][\alpha] \eta = \{ d \mid \forall n \in \omega, d_n \in \mathcal{T}'[-][\alpha] \eta \}. \)

(iii) Let \( \sigma \in T_R \). \( \mathcal{T}'[-][\sigma] \eta = \mathcal{T}'[-][\sigma] \eta \).

Now we give some technical lemmas (3.6–3.8) in order to prove that our interpretation of types is well defined:

**3.6. Lemma.** Let \( \alpha \in T_R \).

(i) \( \mathcal{T}'[-][\alpha] \eta \subseteq \mathcal{T}'[-][\alpha] \eta \)

(ii) If \( d \in \mathcal{T}'[-][\alpha] \eta \) then \( d_n \in \mathcal{T}'[-][\alpha] \eta \)

(iii) \( \mathcal{T}'[-][\alpha] \eta \subseteq \mathcal{T}'[-][\alpha] \eta \).

**Proof.** (i) and (ii) are proved simultaneously by induction on \( n \). The basis is trivial. The inductive step is by case distinction on \( \alpha \). If \( \alpha \) is a type constant, a type variable or \( \Omega \) the proof is immediate. So let \( \alpha = \beta \rightarrow \gamma \).

(i) Let \( d \in \mathcal{T}'[-][\beta \rightarrow \gamma] \eta \) and \( e \in \mathcal{T}'[-][\beta] \eta \). We have, from 3.1(ii-1), \( d(e) = d(en - 1) \) and, by induction hypothesis (ii), \( e_{n-1} \in \mathcal{T}'[-1][\beta] \eta \). Then
\[ d(e) = d(e_{n-1}) \in D_n[\gamma] \eta \subseteq D_n[\gamma] \eta \] by induction hypothesis (i). This implies \( d \in D_n[\gamma] \eta \).

(ii) Let \( d \in D_n[\gamma] \eta \). Take \( e \in D_{n-1}[\beta] \eta \), \( e \in D_n[\beta] \eta \) by induction hypothesis (i) so \( d(e) \in D_n[\gamma] \eta \). By induction hypothesis (ii) and 3.1(ii-2), we have that \((d(e_{n-1}))_{n-1} = d_n(e_{n-1}) = d_n(e) \in D_n[\gamma] \eta \) and this proves \( d_n \in D_n[\beta] \eta \).

(iii) Let \( d \in D_n[\gamma] \eta \); it is enough to prove that for all \( m \in \omega \), \( d_m \in D_m[\gamma] \eta \). To this end we must consider two cases:

- \( n \leq m \); this follows from 3.6(i) and 3.1(i-2) since \( D_n[\gamma] \eta \subseteq D_m[\gamma] \eta \).
- \( m < n \); by repeated applications of 3.6(ii) we have \( d \in D_n[\gamma] \eta \) implies \( d_k \in D_k[\gamma] \eta \) for all \( 0 \leq k \leq n \).

3.7. Lemma. For all \( n \in \omega \):

(i) \( D_n[\gamma] \eta \) is an ideal over \( D_n \);

(ii) \( D_n[\gamma] \eta = (D_n[\gamma] \eta)_n \).

Proof. (i) An easy induction on \( n \) using 3.3(i).

(ii) Easy, using 3.6(iii) for left to right inclusion and 3.5(ii) for the reverse inclusion.

The well-definedness of \( D'[\gamma] \) (for types in \( T_n \)) is given by the following properties.

3.8. Lemma. Let \( x \in T_n \).

(i) \( D'[\gamma] \eta \) is an ideal in \( D \).

(ii) \( D'[\gamma] \eta \to D'[\beta] \eta \).

Proof. (i) Assume \( d \in D'[\gamma] \eta \) and \( e \subseteq d \). For all \( n \geq 0 \), \( e_n \subseteq d_n \) and \( e_n \in D_n[\gamma] \eta \) by 3.7(i), so \( e \in D'[\gamma] \eta \). Now let \( \{d^{(i)}\}_{i \in I} \) be a directed subset of \( D'[\gamma] \eta \) and \( d = \bigsqcup_{i \in I} \{d^{(i)}\} \). For all \( i \in I \) \( d^{(i)} \in D'[\gamma] \eta \) and \( (d^{(i)})_n \in D_n[\gamma] \eta \), with \( \{(d^{(i)})_n\}_{i \in I} \) a directed subset of \( D_n[\gamma] \eta \), and \( \bigsqcup \{(d^{(i)})_n\}_{i \in I} \in D_n[\gamma] \eta \) (by 3.7(i)). Now \( \bigsqcup \{(d^{(i)})_n\}_{i \in I} = d_n \) (by continuity). So we have \( d_n \in D_n[\gamma] \eta \) for all \( n \geq 0 \) and, then, \( d \in D'[\gamma] \eta \).

(ii) Note that, if \( A, B \) are ideals, \( A_n = B_n \) for all \( n \) implies \( A = B \). So we prove that

\[ (D'[\gamma] \eta)_n = (D'[\gamma] \eta \to D'[\beta] \eta)_n \]
for all $n \geq 0$. The case $n = 0$ holds trivially. For $n > 0$ we have

$$(\mathcal{I}'[\alpha \to \beta] \eta)_n = \mathcal{I}'[\alpha] \eta_n \quad \text{(by 3.7(ii))}$$

$$= \mathcal{I}'[\alpha] \eta \xrightarrow{n} \mathcal{I}'[\beta] \eta$$

$$= (((\mathcal{I}'[\alpha] \eta)_{n-1} \xrightarrow{n} (\mathcal{I}'[\beta] \eta)_{n-1}) \quad \text{(by 3.7(ii))}$$

$$= (\mathcal{I}'[\alpha] \eta \to \mathcal{I}'[\beta] \eta)_n \quad \text{(by 3.3(iii)).}$$

From Lemma 3.8 and the fact that $\sigma \sim \tau$ implies $\sigma^* = \tau^*$ we get immediately that the definition of $\mathcal{I}'[-]$ for types in $T_\mu$ is correct.

3.9. Theorem. $\mathcal{I}'[-]: T_\mu \to T_{\text{env}} \to I$ is an I-type interpretation.

We now show that the interpretation defined in 3.5(ii) is indeed the only possible type interpretation (in the sense of Definition 3.4) over $D$ such that $\mu t. t$ is interpreted as $I$.

3.10. Theorem. Let $\mathcal{I}'[-]: T_\mu \to T_{\text{env}} \to I$ be any I-type interpretation. Then

$$\mathcal{I}'[\sigma] \eta = \mathcal{I}'[\sigma] \eta$$

for all types $\sigma \in T_\mu$.

Proof. We show, by induction on $n$, that for all $\sigma \in T_\mu$ $(\mathcal{I}'[\sigma] \eta)_n = (\mathcal{I}'[\sigma] \eta)_n$. This implies the statement immediately. The first step is trivial (note that $A_0 = \{ \bot_D \}$ for all $A \in I$). The induction step is proved by cases on $\sigma$, which has the general form

$$\sigma = \mu t_1 \cdots \mu t_k. \tau,$$

where $k \geq 0$ and $\tau$ does not start with an occurrence of the $\mu$ operator. For $k = 0$, if $\sigma$ is a type constant or a type variable the proof is trivial.

Otherwise, if $\sigma \equiv \rho \to \tau$, we have:

$$(\mathcal{I}'[\rho \to \tau] \eta)_n$$

$$= (\mathcal{I}'[\rho] \eta \to \mathcal{I}'[\tau] \eta)_n \quad \text{by definition of type interpretation}$$

$$= (\mathcal{I}'[\rho] \eta)_n \xrightarrow{n+1} (\mathcal{I}'[\tau] \eta)_n \quad \text{by Lemma 3.3(iii)}$$

$$= (\mathcal{I}'[\rho] \eta)_n \xrightarrow{n+1} (\mathcal{I}'[\tau] \eta)_n \quad \text{by induction hypothesis}$$

$$= (\mathcal{I}'[\rho] \eta \to \mathcal{I}'[\tau] \eta)_{n+1} = (\mathcal{I}'[\rho \to \tau] \eta)_{n+1}.$$
which is either a variable or a type constant or an arrow type, and we are lead back to the previous cases. Else we have $\sigma \sim \mu.t.t$, in which case $(\mathcal{I}[\mu.t.t]\eta)_n = I_n = (\mathcal{I}[\mu.t.t]\eta)_n$. 

3.11. Remark. As a consequence of the preceding theorem we have that our type interpretation coincides with that defined in (MacQueen, Plotkin, and Sethi, 1986) if restricted to what they call well formed type expressions. These are all type expressions not containing subtypes of the form $\mu.t_1, \ldots \mu.t_n$. In fact, it is easy to verify that the interpretation of such types is independent of $I$ and that the interpretation of types given in (MacQueen, Plotkin, and Sethi, 1986) is a type interpretation in our sense (where only the set of well formed type expressions is considered). The following argument may clarify the relations between their approach and the present one. Let $\mu.t.\sigma$ be a well formed type expression. MacQueen, Plotkin, and Sethi define the interpretation of $\mu.t.\sigma$ as

$$\mathcal{I}[\mu.t.\sigma]\eta = \text{fix}(\Gamma),$$

where $\Gamma = \lambda I \in \mathcal{I}. \mathcal{I}[\sigma]\eta[t \mapsto I]$ and $\text{fix}(\Gamma)$ is the unique fixed point of the contractive function $I : \mathcal{I} \to \mathcal{I}$ (with respect to the metrics they define on $\mathcal{I}$). By the Banach fixed point theorem $\text{fix}(\Gamma)$ is defined by $\lim_{n \to \infty} f^n(J)$ where $J$ is an arbitrary ideal of $D$. It is easy to check (see also Amadio, 1989) that

$$\mathcal{I}_n[\mu.t.\sigma]\eta = (\Gamma^n(J))_n,$$

i.e., the $n$th iteration of $\Gamma$ is equal to the interpretation of $\mu.t.\sigma$ up to the $n$th level of approximation.

Our construction, although more syntactical in nature, is perhaps more natural and can be more tractable in applications, as the following proof of the completeness of $\approx$ for semantically type equality and the Completeness Theorem in Section 4.

The following corollary is an immediate consequence of Theorem 3.10.

3.12. Corollary. In any $l$-type interpretation $\mathcal{I}[\cdot]$, $\sigma \approx \tau$ implies $\mathcal{I}[\sigma]\eta = \mathcal{I}[\tau]\eta$.

In the rest of this section we shall prove that the relation $\approx$ gives a complete characterization of semantic equality of types provided we assume that $D$ is a domain in the usual sense.

A c.p.o. is consistently complete if every subset which has an upper bound has a least upper bound. An element $d$ of a c.p.o. $D$ is finite if whenever $d \subseteq \bigcup X$ (where $X$ is a directed subset of $D$) we have $d \subseteq x$ for some $x \in X$. $D$ is algebraic if for all $d \in D$ the set $\{a \mid a \subseteq d \text{ and } a \text{ is finite}\}$ is directed and has $d$ as its least upper bound. Following (Scott, 1982) we define a
domain as a consistently complete algebraic c.p.o. From now on we will assume that D is a domain.

If d, e are finite elements of D then \(d \Rightarrow e\) denotes the (continuous) step function defined by

\[
(d \Rightarrow e)(x) = \begin{cases} 
  d & \text{if } d \subseteq x \\
  e & \text{otherwise} 
\end{cases}
\]

\((d \Rightarrow e)\) is a finite element of \([D \rightarrow D]\). Moreover if d, e are finite elements of \(D_n\) then \((d \Rightarrow e)\) is a finite element of \(D_{n+1}\). Observe that, in general, \(d_n\) need not be a finite element of \(D\) nor of \(D_n\).

Note that if \(A\) and \(B\) are two ideals over a domain \(D\), \(A \neq B\) implies that there exists a finite element \(d \in A\) such that \(d \notin B\) or vice versa.

3.13. LEMMA. Let \(D\) be a domain.

(i) Let \(A_n, B_n, A'_n, B'_n\) be ideals in \(1_n\). Then \(A_n \overset{n+1}{\longrightarrow} B_n = A'_n \overset{n+1}{\longrightarrow} B'_n\) implies \(A_n = A'_n\) and \(B_n = B'_n\).

(ii) If \(A, A', B, B'\) are ideals in \(D\), then \(A \rightarrow B \Rightarrow A' \rightarrow B'\) implies \(A = A'\) and \(B = B'\).

Proof: (i) If \(A_n \neq A'_n\) then there must be a finite element \(d \in A_n\) such that \(d \notin A'_n\) (or vice versa). Taking an arbitrary finite element \(e\) of \(D\) which is not an element of \(B_n\) (note that at least \(e \notin B_n\)) we obtain \((d \Rightarrow e) \in A'_n \overset{n+1}{\longrightarrow} B'_n\) but \((d \Rightarrow e) \notin A_n \overset{n+1}{\longrightarrow} B_n\), against the hypothesis.

Similarly, assume \(B_n \neq B'_n\) and take a finite \(e \in D_n\) such that \(e \in B_n\) but \(e \notin B'_n\) (or vice versa). The constant function \(\lambda x. e\) is an element of \(A_n \overset{n+1}{\longrightarrow} B_n\) but not of \(A'_n \overset{n+1}{\longrightarrow} B'_n\).

(ii)

\[
A \rightarrow B = A' \rightarrow B' \Rightarrow \forall n. (A \rightarrow B)_{n+1} = (A' \rightarrow B')_{n+1}
\]

\[
\Rightarrow \forall n. A_n \overset{n+1}{\longrightarrow} B_n = A'_n \overset{n+1}{\longrightarrow} B'_n
\]

(by Lemma 3.3(iii))

\[
\Rightarrow \forall n. A_n = A'_n\text{ and }B_n = B'_n
\]

(by point (i))

\[
\Rightarrow A = A'\text{ and }B = B'.
\]

Let now \(\alpha, \beta \in T_R\). We define a family of relations \(\approx_n \subseteq T_R \times T_R\) inductively on \(n \in \omega\) in the following way:

1. \(\alpha \approx_0 \beta\) for all \(\alpha, \beta \in T_R\).

\(\approx_{n+1}\) is the smallest relation over \(T_R\) such that the following conditions hold:

2. \(\kappa \approx_{n+1} \kappa\) for each type constant \(\kappa\).
3. \( \Omega \approx_{n+1} \Omega \).
4. \( t \approx_{n+1} t \) for each type variable \( t \).
5. \( \alpha_1 \rightarrow \alpha_2 \approx_{n+1} \beta_1 \rightarrow \beta_2 \) if \( \alpha_1 \approx_n \beta_1 \) and \( \alpha_2 \approx_n \beta_2 \).

Clearly we have that \( \alpha = \beta \) iff for all \( n \in \omega \), \( \alpha \approx_n \beta \).

3.14. **Lemma.** Let \( D \) be a domain and \( \alpha, \beta \in T_R \). Then \( \alpha \approx_n \beta \) iff for all \( \eta \) and for all \( I \in \mathbf{I} \), \( \mathcal{I}^I_{n}[\alpha] \eta = \mathcal{I}^I_{n}[\beta] \eta \).

**Proof.** By induction on \( n \). If \( n = 0 \) the proof is immediate. The induction step is by cases on \( \alpha \). If \( \alpha \) is a type constant, type variable, or \( \Omega \) the proof is trivial. So let \( \alpha = \alpha_1 \rightarrow \alpha_2 \). If \( \alpha \approx_{n+1} \beta \) then \( \beta \) must be of the shape \( \beta_1 \rightarrow \beta_2 \) (remark that the equality of the interpretations must hold for all type environments \( \eta \) and all choices of \( I \)) and it follows from the definition that \( \alpha_1 \rightarrow \alpha_2 \approx_{n+1} \beta_1 \rightarrow \beta_2 \) implies \( \alpha_1 \approx_n \beta_1 \) and \( \alpha_2 \approx_n \beta_2 \). Then we have

\[
\mathcal{I}^I_{n+1}[\alpha_1 \rightarrow \alpha_2] \eta = \mathcal{I}^I_{n+1}[\beta_1 \rightarrow \beta_2] \eta
\]

\[
\iff \mathcal{I}^I_{n}[\alpha_1] \eta \approx_{n+1} \mathcal{I}^I_{n}[\alpha_2] \eta = \mathcal{I}^I_{n}[\beta_1] \eta \approx_{n+1} \mathcal{I}^I_{n}[\beta_2] \eta
\]

\[
\iff \mathcal{I}^I_{n}[\alpha_1] \eta = \mathcal{I}^I_{n}[\beta_1] \eta
\]

and \( \mathcal{I}^I_{n}[\alpha_2] \eta = \mathcal{I}^I_{n}[\beta_2] \eta \) (using Lemma 3.13(i))

\[
\iff \alpha_1 \approx_n \beta_1 \text{ and } \alpha_2 \approx_n \beta_2 \quad \text{(by induction hypothesis)}
\]

\[
\iff \alpha_1 \rightarrow \alpha_2 \approx_{n+1} \beta_1 \rightarrow \beta_2. \]

The completeness of \( \approx \) with respect to the semantic equality of types is stated by the following theorem.

3.15. **Theorem.** Let \( D \) be a domain.

(i) Let \( \alpha, \beta \in T_R \). \( \alpha = \beta \) iff for all \( \eta \in T_{env} \) and for all \( I \in \mathbf{I} \), \( \mathcal{I}^I[I] \eta = \mathcal{I}^I[I] \eta \).

(ii) Let \( \sigma, \tau \in T_\mu \). \( \sigma \approx \tau \) iff for all \( \eta \in T_{env} \) and for all \( I \in \mathbf{I} \), \( \mathcal{I}^I[I] \eta = \mathcal{I}^I[I] \eta \).

**Proof.** (i) \( \alpha = \beta \iff \forall n \in \omega \) \( \alpha \approx_n \beta \iff \forall \eta, I, \) and \( n \), \( \mathcal{I}^I_n[I] \eta = \mathcal{I}^I_n[I] \eta \) (by Lemma 3.14) \( \iff \forall \eta \) and \( I \), \( \mathcal{I}^I[I] \eta = \mathcal{I}^I[I] \eta \).

(ii) From (i) and the fact that \( \sigma \approx \tau \iff (\sigma)^* = (\tau)^* \).

A consequence of Remark 3.11 is that Theorem 3.15 holds also for the well-formed type expressions of (MacQueen, Plotkin, and Sethi, 1986) w.r.t. the metric interpretation.

3.16. **Remark.** The definition of approximable application can be
defined in a more general framework in which we assume only that 
\([D \to D]\) is a retract of \(D\); i.e., there exist two maps \(F: D \to [D \to D]\) and 
\(G: [D \to D] \to D\) such that \(F \circ G = \text{Id}_{[D \to D]}\) (this is obviously the case of a 
domain satisfying \([\ast]\) where we have \(F = \text{out}_{[D \to D]}\), the projection func-
tion of \(D\) in \([D \to D]\), and \(G = \text{in}_{D}\)). The condition that \([D \to D]\) is a 
retract of \(D\) characterizes a wide class of topological models of the pure 
\(\lambda\)-calculus (see Barendregt, 1984, Chap. 5.6). In the case of pure \(\lambda\)-models 
we must drop out point 4 of Definition 3.1(i) which becomes meaningless, 
but observe that the existence of basic types is not essential to the definition 
and construction of type interpretation. Examples of models of pure 
\(\lambda\)-calculus with approximable application are the models \(D_A\) of (Engeler, 
1981) and the filter model of (Barendregt, Coppo, and Dezani Ciancaglini, 
1983). All the results of this section hold also for \(\lambda\)-models with 
approximable application.

Remark that the standard \(D_{\infty}\) (Scott, 1972) constructed starting from an 
initial c.p.o. \(D_0\) is not, strictly speaking, a domain with approximable 
application. In fact point (i-1) of Definition 3.1 fails, due to the fact that 
\(D_0\) is different from \(\{\bot\}\), so there are elements \(d \in D_\infty\) such that \(d_0 \neq \bot\). In 
this case we have that \(\mathcal{I}'[\_\_]\) is still a type interpretation if we relativize 
the construction of type interpretation to ideals \(I\) satisfying \(I_0 = \{\bot\}\). In 
general, Theorems 3.10, 3.14, and 3.15 are no longer true. For example, 
\(\mathcal{I}'[\mu t.t \to t] \eta\) turns out to be a proper subset of \(D\) (containing the interpret-
atations of all closed terms) but, since \(D_{\infty} \cong [D_{\infty} \to D_{\infty}]\), there is 
another type interpretation \(\mathcal{I}'[\_\_]\) such that \(\mathcal{I}'[\mu t.t \to t] \eta = D_{\infty}\).

4. A Complete System

In this section we study the completeness of type assignment with respect 
to the semantics presented in Section 3. The basic system of Definition 2.3 
is effective but not complete. So we introduce a more powerful (but non-
effective) system which will be proved to be complete with respect to the 
interpretation of types as ideals.

First we give a denotational semantics to our language of expressions in 
a domain \(D\) with approximable application, satisfying \(D \cong At + W + 
[D \to D]\). We assume in the proof below that the sum of domains in the 
equation is interpreted as a disjoint sum (we remark, however, that the whole 
completeness proof can be adapted as well to an equation which uses 
the coalesced sum).

Let \(C: C \to D\) be an interpretation of the constants of the language. A 
term (environment) is any function \(\rho: V \to D\). Let Env be the set of term 
environments. If \(d \in D\), we use the notation \(\rho[x \mapsto d]\) to denote the 
environment which is equal to \(\rho\) except that \(\rho[x \mapsto d](x) = d\).
4.1. **Definition.** (i) \( \bullet: \[D \rightarrow [D \rightarrow D]\] \) (application in \( D \)) is defined by
\[
d \bullet e = \begin{cases} 
\bot & \text{if } d = \bot \\
\langle d \rangle & \text{if } d \in [D \rightarrow D] \\
? & \text{otherwise.}
\end{cases}
\]

(ii) The denotational semantics of the type free language is given by a function
\[
\llbracket \ \rrbracket : A \rightarrow (\text{Env} \rightarrow D)
\]
defined by
\[
\llbracket x \rrbracket \rho = \rho(x) \\
\llbracket c \rrbracket \rho = \mathcal{C}(c) \\
\llbracket \lambda x. M \rrbracket \rho = \lambda \langle d \rangle \in D. \llbracket M \rrbracket \rho[x \mapsto d] \\
\llbracket (M N) \rrbracket \rho = \llbracket M \rrbracket \rho \bullet \llbracket N \rrbracket \rho.
\]

Observe that \( \bullet \) represents a call-by-name notion of application which preserves the interpretation of terms under \( \beta \)-convertibility.

Since \( + \) represents disjoint sum we have \( \lambda v. \bot \neq \bot \) as elements of \( D \).

This implies, for instance, that \( \llbracket \lambda x. M \rrbracket \rho \neq \llbracket M \rrbracket \rho \) even if \( M \) is an unsolvable term. In particular, in our interpretation, the terms which are interpreted as \( \bot \) (in all environments) are exactly the unsolvable terms of order 0, i.e., all unsolvable terms \( M \) such that \( M \) does not reduce to a term of the form \( \lambda x. N \) (Barendregt, 1984. Chap. 17.3).

We can now define the semantics of the sequents of the type assignment system in the model \( D \).

4.2. **Definition.** (i) A term environment \( \rho \) and a type environment \( \eta \) satisfy a basis \( B \) in the type interpretation \( \mathcal{T} \) (notation \( \mathcal{T}, \rho, \eta \vdash B \)) if for all \( x : \sigma \in B \rho(x) \in \mathcal{T}^{\llbracket \sigma \rrbracket} \eta \).

(ii) \( B \vdash M : \tau \) iff for all \( I \in \mathcal{I} \) and for all type interpretation \( \mathcal{T}, \rho, \eta \vdash B \) implies \( \llbracket M \rrbracket \rho \in \mathcal{T}^{\llbracket \tau \rrbracket} \eta \).

The soundness of the type assignment rules can be shown by a simple induction on derivations.

4.3. **Theorem (Soundness).** \( B \vdash_{\mu} M : \sigma \) implies \( B \vdash M : \sigma \).

This theorem, in conjunction with Theorem 3.9, entails a semantical version of Remark 2.6(ii), because \( ? \notin \mathcal{T}^{\llbracket \sigma \rrbracket} \eta \) for any \( \sigma \in T_{\mu} \) and \( \eta \in T_{\text{env}} \).
The converse of Theorem 4.3, i.e., the completeness of the type assignment rules, fails for several reasons. Observe that we aim at a strong form of completeness, relative to our specific model $D$ for the base language. In this case even $\vdash_{\mu,\text{Eq}}$ is not complete, because equality in the model is stronger than convertibility (however, a completeness theorem for $\vdash_{\mu,\text{Eq}}$ with respect to validity of typing statements in all models of the base language can be proved by a straightforward extension of the technique used in (Hindley, 1983)).

As an example of how completeness fails for $\vdash_{\mu,\text{Eq}}$ take $A_3 \equiv \lambda x. \text{xxx}$: we have $[[A_3 A_3]] = \bot$ (because it is an unsolvable term of order 0) so, for all types $\sigma$, $[[A_3 A_3]] \in \mathcal{J}[[\sigma]]$ (i.e., $\vdash (A_3 A_3) : \sigma$). But we have that the p.t.s. of $(A_3 A_3)$ (and of any term convertible to it) is $\mu t. t \to t$, so $(A_3 A_3) : \sigma$ cannot be derived in $\vdash_{\mu,\text{Eq}}$ (nor, obviously, in $\vdash_{\mu}$) for all $\sigma$ different from $\mu t. t \to t$.

This incompleteness of $\vdash_{\mu,\text{Eq}}$ cannot be overcome by addition of a rule which permits assignment of every type to unsolvable terms of order 0 (which would be sufficient to deal with the previous example). Consider the term $G \equiv \lambda f. \theta f \theta f S$, where $\theta f \equiv \lambda x y. f(y \text{III} x S)$, $S \equiv \lambda x y z. x z(y z)$ and $I \equiv \lambda x. x$. Observe that $GM =_\beta M(GM)$, so the term $G$ is a fixed point combinator. It is solvable and equal, in our model, to $Y = \text{fix} \lambda f. (\lambda x. f(xx))(\lambda x. f(xx))$, but the only type assignable to $G$ is also in this case $\mu t. t \to t$, while $\vdash G : (t \to t) \to t$.

We now introduce an extension of the system which will be proved complete with respect to the interpretation of types in $D$. In order to formulate this extension, we need the notion of approximant of a term, following (Wadsworth, 1976). We will give here only the basic definitions and properties: for more details we refer to (Wadsworth, 1976) or (Mosses and Plotkin, 1987).

We add a new constant $\bot$ to the language of $\lambda$-terms (whose intended interpretation is $\bot$). Let $A_\bot$ be the set of terms so obtained ($\lambda$-$\bot$-terms).

4.4. DEFINITION. (i) Let $M \in A$. The direct approximant of $M$, $\omega(M) \in A_\bot$, is defined inductively on $M$ in the following way:

1. $\omega(c) = c$ if $c$ is a constant
2. $\omega(x) = x$ if $x$ is a variable
3. $\omega(\lambda x. M) = \lambda x. \omega(M)$
4. $\omega(x M_1 \cdots M_k) = x \omega(M_1) \cdots \omega(M_k)$ ($k > 0$)
5. $\omega(c M_1 \cdots M_k) = c \omega(M_1) \cdots \omega(M_k)$ ($k > 0$)
6. $\omega((\lambda x. N) M_1 \cdots M_k) = \bot$ ($k > 0$).

(ii) Let $M \in A$. The set of approximants of $M$ is given by

$$\text{APP}(M) = \{ A \in A_\bot \mid M \xrightarrow{\Delta} N \text{ and } A = \omega(N) \}.$$
\( \lambda \)-\( \perp \)-terms are interpreted assuming \( \llbracket \perp \rrbracket \rho = \perp \). In domains with approximable application the following Approximation Theorem holds (see Wadsworth, 1976, and, for a more general setting, Longo, 1983).

4.5. THEOREM (Approximation Theorem).

\[
\llbracket M \rrbracket \rho = \bigcup \{ \llbracket A \rrbracket \rho \mid A \in \text{APP}(M) \}.
\]

Theorem 4.5 suggests immediately the following extension of \( \vdash _\mu \).

4.6. DEFINITION. The type assignment system \( \vdash _\mu , \nu \) is obtained by adding to the system \( \vdash _\mu \) of Definition 2.3 the following two rules:

\[
\begin{align*}
B \vdash _\mu , \nu \perp : \sigma & \quad (\perp \text{ for all types } \sigma) \\
B \vdash _\mu , \nu A : \sigma \text{ for all } A \in \text{APP}(M) & \quad B \vdash _\mu , \nu M : \sigma \quad (C).
\end{align*}
\]

Axiom \( (\perp) \) is motivated by the fact that \( \perp \) (the interpretation of \( \perp \)) belongs to all types, and rule \( (C) \) by the Approximation Theorem 4.5. Continuing the example above, the approximants of \( G \) are of the form 

\( \lambda f . f^n(\perp) : (t \to t) \to t \),

so also \( G \) has type \( (t \to t) \to t \) by rule \( (C) \).

In fact, with some reasonable assumptions on the choice of the constants (given below), \( \vdash _\mu , \nu \) turns out to be complete with respect to the type interpretation introduced in Section 3. Note that rule \( (C) \) has an infinitary nature. Indeed, \( \vdash _\mu , \nu \) is \( \Pi^0_1 \)-complete. However, since also \( \models \) is \( \Pi^0_1 \)-complete (note that a term belongs to the interpretation of all types iff it is unsolvable of order 0, and this notion is \( \Pi^0_1 \)), there is no way of finding a simpler complete system.

4.7. THEOREM. \( B \vdash _\mu , \nu M : \sigma \) iff \( B \models M : \sigma \).

The implication from left to right can be proved by a straightforward (transfinite) induction on derivations. The rest of this section is devoted to the proof of the converse implication (completeness). As a corollary of this proof we have a partial completeness result for \( \vdash _\mu \), i.e., that \( \vdash _\mu \) is complete with respect to terms in normal form.

4.8. COROLLARY. If \( M \) is in normal form then \( B \vdash _\mu M : \sigma \) iff \( B \models M : \sigma \).

As remarked before, the completeness of theorem can be adapted also to
the case that $+$ is the coalesced sum. In this case, however, we have that $\lambda v. \bot = \bot$ (as elements of $D$) and then all unsolvable terms are equated in the model. In this case to get a complete system we have to modify the notion of approximant of a term by identifying $\lambda x_1 \cdots \lambda x_n. \bot$ with $\bot$.

4.9. Remark. If we had defined the notion of type semantics assuming $I = \{\bot\}$, without considering $I$ as a parameter in Definition 3.4, we would have, for instance, $x:\mu. t \models x: \sigma$ for all types $\sigma$. In this case, then, the completeness of $\vdash_{\mu,C}$ fails since we have no way of deriving $x: \sigma$ assuming $x:\mu. t$. A complete system might be obtained by introducing a formal relation of inclusion $\leq$ between types such that $\mu. t \leq \sigma$ for all types $\sigma$ (see Mitchell, 1988, for a discussion of type assignment with inclusion). In the present context, however, we believe our choice to be more general, as we do not see any other reason to assume $\mu. t \leq \sigma$.

The same problem arises, indeed, with any choice of the ideal $I$.

Proof of the Completeness Theorem

To prove Theorem 4.7 we use (a variant of) the technique introduced in (Coppo, 1984). The main idea is to define, for each type $\alpha$ and integer $n$, a set of values $T^{\alpha,n} \subseteq D_n$ which completely characterize the behaviour of the elements of type $\alpha$ in $D_n$. For technical reasons it is useful to consider, for a while, only regular types without occurrences of $\Sigma$ and of type variables. Let $T_{\mu} \subseteq T_R$ denote the set of all such types. $T_{\mu}$ contains all and only the unfoldings of closed types in $T_{\mu}$ which do not have subtypes equivalent of $\mu. t$. For instance $(\mu.(t \rightarrow \text{int}))^* \in T_{\mu}$. To simplify notations we will denote the interpretation of types in $T_{\mu}$ omitting the type environment and the ideal $I$ (so we write simply $\mathcal{I}[\alpha], \mathcal{I}_n[\alpha]$ instead of $\mathcal{I}^\eta[\alpha], \mathcal{I}_n^\eta[\alpha])$. Note that the environment $\eta$ and the choice of $I$ are irrelevant to the interpretation of types in $T_{\mu}$.

As remarked before, we must put some restrictions on the interpretation of constants. Let $v$ be a basic or simply functional type (a finite type $v$ is a simply functional type if $v = \kappa \rightarrow v'$, where $\kappa$ is a type constant and $v'$ is either a basic or a simply functional type). A value $v \in \mathcal{I}[v]$ strongly belongs to $v$ (notation $v \in_s \mathcal{I}[v]$) iff:

1. $v \neq \bot$
2. if $v = \kappa$ then $v \in \mathcal{I}[\kappa]$
3. if $v = \kappa \rightarrow v'$ then $\forall v' \notin \mathcal{I}[\kappa] \therefore v(v') = ?$ and $\forall v' \in_s \mathcal{I}[\kappa] v(v') \in_s \mathcal{I}[v'].$

An interpretation of constants $\mathcal{C}$ is well behaved iff for all $c \in C \tau_c(c)$ is either a basic or a simply functional type and $\mathcal{C}(c) \in_s \mathcal{I}[\tau_c(c)]$. 


We assume, from now on, that the interpretation of constants is well behaved.

We need some auxiliary definitions.
Define \( d^\kappa \) and \( d^- \) by

\[
d^\kappa(v) =
\begin{cases}
\bot & \text{if } v = \bot \\
Id_D & \text{if } v \in \mathcal{F}[^\kappa] \text{ and } v \neq \bot \\
? & \text{otherwise}
\end{cases}
\]

\[
d^{-}(v) =
\begin{cases}
\bot & \text{if } v = \bot \\
Id_D & \text{if } v \in [D \rightarrow D] \text{ and } v \neq \bot \\
? & \text{otherwise}
\end{cases}
\]

\( d^\kappa \) and \( d^- \) characterize, respectively, \( K \) (the component of \( D \) which interprets \( \kappa \)) and \([D \rightarrow D]\) in the sense that they yield \(?\) whenever applied to an element of \( D \) (\( \neq \bot \)) which does not belong to \( K \) or to \([D \rightarrow D]\). Note that \( d^\kappa \) and \( d^- \) are continuous and internally representable.

If \( A, B \subseteq D \) let \( A \cdot B \) and \( \lambda v. A \) denote, respectively, the sets \( \{a \cdot b \mid a \in A, b \in B\} \) and \( \{\lambda v. a \mid a \in A\} \). Observe that, if \( A \subseteq [D \rightarrow D] \), \( A(B) = A \cdot B = \{a(b) \mid a \in A, b \in B\} \).

4.10. DEFINITION. Let \( \alpha \in T^\omega \). For \( n > 0 \), the sets \( T^{\alpha,n} \subseteq D_n \) and \( A^{\alpha,n} \subseteq [D \rightarrow D] \) are defined, by induction on \( n \), in the following way:

1. \( T^{\beta \cdot \gamma,1} = \{\lambda v. \bot\} \) for all \( \beta, \gamma \in T^\omega \).
2. \( A^{\beta \cdot \gamma,1} = \{d^-\} \) for all \( \beta, \gamma \in T^\omega \).
3. \( T^{\kappa,n} = \{v_\kappa\} \) where \( v_\kappa \) is any value that strongly belongs to \( \kappa \).
4. \( A^{\kappa,n} = \{d^\kappa\} \).
5. \( T^{\beta \cdot \gamma,n+1} = (\lambda v. A^{\beta,n}(v) \cdot T^{\gamma,n})_{n+1} \cup (\lambda v. T^{\gamma,n})_{n+1} \).
6. \( A^{\beta \cdot \gamma,n+1} = \lambda v. A^{\gamma,n}(v \cdot T^{\beta,n}) \).

In the next lemma it is proved that each element of \( T^{\alpha,n} \) belongs to (the \( n \)th level of) the interpretation of \( \alpha \), and \( A^{\alpha,n} \) is a set of functions (representable in \( D \)) that, in some sense, characterize type \( \alpha \) (up to the \( n \)th level of approximation) in the same way as \( d^- \) and \( d^\kappa \) do for \([D \rightarrow D]\) and \( K \).

4.11. LEMMA. (i) \( T^{\alpha,n} \subseteq \mathcal{F}[\alpha] \) for \( n > 0 \).

(ii) For all \( v \in \mathcal{F}[\alpha] \) and \( n > 0 \) \( A^{\alpha,n}(v) \subseteq \{\bot, Id_D\} \).

(iii) For all \( v \in D \) and \( n > 0 \) \( A^{\alpha,n}(v) \subseteq \{\bot, Id_D, ?\} \).

Proof. (i) and (ii) are proved simultaneously, by induction on \( n \). The first step \( (n = 1) \) is trivial.
The inductive step is by cases on $\alpha$. If $\alpha = \kappa$ (a type constant) the proof is again trivial.

Consider the case $\alpha \equiv \alpha' \rightarrow \alpha''$.

(i) We have $T^{\alpha' \rightarrow \alpha'' \cdot n+1} = A \cup B$, where $A = (\lambda v. A^{\alpha' \cdot n}(v) \cdot T^{\alpha'' \cdot n})_{n+1}$ and $B = (\lambda v. T^{\alpha'' \cdot n})_{n+1}$. We have immediately $B \subseteq \mathcal{J}_{n+1}[\alpha' \rightarrow \alpha'']$ by induction hypothesis (i). As for $A$ let $a_n \in \mathcal{J}_n[\alpha']$. By 3.1(ii-3) we have:

$$A(a_n) = (A^{\alpha' \cdot n}(a_n) \cdot T^{\alpha'' \cdot n})_n \subseteq T^{\alpha' \rightarrow \alpha'' \cdot n} \cup \{ \bot \} \subseteq \mathcal{J}_n[\alpha'']$$

using induction hypothesis (ii) and the fact that $(T^{\alpha'' \cdot n})_n = T^{\alpha'' \cdot n}$.

(ii) Let $v \in \mathcal{J}[\alpha' \rightarrow \alpha'']$. We have $A^{\alpha' \rightarrow \alpha'' \cdot n+1}(v) = A^{\alpha' \cdot n}(v \cdot T^{\alpha'' \cdot n})$, and $v \cdot T^{\alpha'' \cdot n} \subseteq \mathcal{J}[\alpha'']$ by induction hypothesis (i) (note that if $v \neq \bot$ then $v \cdot T^{\alpha'' \cdot n} = v(T^{\alpha'' \cdot n})$). Then apply induction hypothesis (ii).

(iii) By induction on $n$. The first step is trivial. For the induction step observe that $A^{\alpha' \rightarrow \alpha'' \cdot n+1}(v) = A^{\alpha' \cdot n}(v \cdot T^{\alpha'' \cdot n})$ and apply induction hypothesis.

$T^{\alpha, n}$ and $A^{\alpha, n}$, indeed, characterize type $\alpha$ up to level $n$ in the sense that, for example, $A^{\alpha, n}(T^\beta, n)$ does not contain the error element $?\alpha$ iff $\alpha$ and $\beta$ are equal up to level $n$.

4.12. LEMMA. For all $n > 0$:

(i) $T^{\alpha, n} \subseteq \mathcal{J}_n[\beta] \Rightarrow \mathcal{J}_n[\alpha] = \mathcal{J}_n[\beta]$.

(ii) $? \notin A^{\alpha, n}(T^\beta, n)$ implies $\mathcal{J}_n[\alpha] = \mathcal{J}_n[\beta]$ and $A^{\alpha, n}(T^\beta, n) = \{Id_D\}$.

(iii) If $? \notin T^{\alpha' \rightarrow \beta \cdot n+1} \cdot T^{\gamma, n}$ then $\mathcal{J}_n[\alpha] = \mathcal{J}_n[\gamma]$ and $T^{\alpha' \rightarrow \beta \cdot n+1} \cdot T^{\gamma, n} = T^\beta, n$.

Proof. By simultaneous induction on $n$. The first step ($n = 1$) is trivial. In (i) and (ii) the only interesting case in the induction step is $\alpha = \alpha' \rightarrow \alpha''$. Observe that, in this case, we must have $\beta = \beta' \rightarrow \beta''$.

(i) Take $(\lambda v. A^{\alpha', n}(v) \cdot T^{\alpha'' \cdot n})_{n+1} \subseteq T^{\alpha' \rightarrow \alpha'' \cdot n+1} \subseteq \mathcal{J}_{n+1}[\beta' \rightarrow \beta'']$. Since $T^{\beta, n} \subseteq \mathcal{J}_n[\beta']$ we have $A^{\alpha', n}(T^\beta, n) \cdot T^{\alpha'' \cdot n} \subseteq \mathcal{J}_n[\beta'']$ which implies $? \notin A^{\alpha', n}(T^\beta, n)$.

By induction hypothesis (ii) we have $\mathcal{J}_n[\alpha'] = \mathcal{J}_n[\beta']$ and $A^{\alpha', n}(T^\beta, n) \cdot T^{\alpha'' \cdot n} = \mathcal{J}_n[\beta'']$. By induction hypothesis (i) then $\mathcal{J}_n[\alpha''] = \mathcal{J}_n[\beta'']$ which implies $\mathcal{J}_{n+1}[\alpha' \rightarrow \alpha''] = \mathcal{J}_{n+1}[\beta' \rightarrow \beta'']$.

(ii) $A^{\alpha' \rightarrow \alpha'' \cdot n+1}(T^\beta \rightarrow \beta' \cdot n+1) = A^{\alpha', n}(T^\beta \rightarrow \beta' \cdot n+1) \cdot T^{\alpha'' \cdot n}$. The proof follows easily using induction hypothesis (ii) and (iii).

(iii) $T^\alpha \rightarrow \beta, n+2 \cdot T^\gamma, n+1 - A^{\alpha, n+1}(T^\gamma, n+1) \cdot T^\beta, n$. By point (ii) we have $\mathcal{J}_n[\alpha] = \mathcal{J}_n[\gamma]$ and $A^{\alpha, n+1}(T^\gamma, n+1) = \{Id_D\}$ which implies immediately $A^{\alpha, n+1}(T^\gamma, n+1) \cdot T^\beta, n = T^\beta, n$. 1
$\forall \sigma \rightarrow \beta.n$ can be safely applied to $T^{\sigma,m}$ for any $m$, in the following sense.

4.13. **Lemma.** $T^{\sigma,\beta,n}(T^{\sigma,m}) \subseteq T^{\beta,n} \cup \{\bot\} \ (n > 0)$.

**Proof.** Observe that $T^{\sigma,\beta,n}(T^{\sigma,m}) = \Delta^{\sigma,n}((T^{\sigma,m})_{\mu}) \cdot T^{\beta,n} \cup T^{\beta,n}$ and apply 4.11(i) and (ii).

Let now $T^{\sigma,\beta}_{\mu}$ be the subset of $T^{\sigma,\beta}$ without occurrences of free variables and of subtypes equivalent to $\mu \cdot t$. $T^{\sigma,\beta}_{\mu}$ is the subset of all types $\sigma \in T^{\sigma,\beta}$ such that $\sigma^* \in T^{\sigma,\beta}_{\mu}$. For $\sigma \in T^{\sigma,\beta}_{\mu}$, define $T^{\sigma,\beta,\mu} = T^{\sigma,\beta,\mu}$. Now define

$$T^{\sigma} = \bigcup_{n \in \omega} T^{\sigma,n+1} \cup \{\bot\}$$

$$\Delta^{\sigma} = \bigcup_{n \in \omega} \Delta^{\sigma,n+1} \cup \{\bot\}.$$

The properties of $T^{\sigma,\beta,\mu}$, $\Delta^{\sigma,\beta,\mu}$ extend straightforwardly, in the limit, to $T^{\sigma,\beta}$, $\Delta^{\sigma,\beta}$.

4.14. **Lemma.** Let $\sigma, \tau, \rho \in T^{\sigma,\beta}_{\mu}$.

(i) $T^{\sigma} \subseteq \exists[\tau]$ implies $\sigma \approx \tau$.

(ii) $\not\exists \Delta^{\sigma}(T^{\sigma})$ implies $\sigma \approx \tau$ and $\Delta^{\sigma}(T^{\sigma}) = \{\bot, \text{Id}_D\}$.

(iii) $\not\exists \rho$ $T^{\sigma,\beta} \cdot T^{\rho,\beta}$ implies $\sigma \approx \rho$ and $T^{\sigma,\beta} \cdot T^{\rho,\beta} = T^{\tau}$.

**Proof.** (i) Note that $T^{\sigma} \subseteq \exists[\tau]$ implies $T^{\sigma,\beta,\mu} \subseteq \exists_{\mu}[\tau^*]$ for all $n > 0$. Then, by 4.12(i), we have $\exists_{\mu}[\tau^*] = \exists_{\mu}[\tau^*]$ for all $n > 0$ and, by 3.15(ii), $\sigma \approx \tau$. The proofs of (ii) and (iii) are similar using 4.11, 4.12, and 4.13.

Let $B^{\sigma}$ denote a basis such that all types occurring in it belong to $T^{\sigma,\beta}_{\mu}$. We define the family of (term) environments $\rho_{B^\sigma}$ as:

$$\rho_{B^\sigma} = \{\rho \mid \rho(x) \in T^{\sigma} \text{ if } x : \sigma \in B^{\sigma}, \text{ else } \rho(x) = ?\}$$

Let $\vdash_{\mu,\bot}$ be the system obtained by eliminating rule (C) from $\vdash_{\mu, C}$.

4.15. **Lemma.** Let $A$ be an approximate normal form.

(i) $[A]_{\rho_{B^\sigma}} \subseteq \exists[\sigma]$ implies $B^{\sigma} \vdash_{\mu,\bot} A : \sigma$.

(ii) $\not\exists \Delta^{\sigma}([A]_{\rho_{B^\sigma}})$ implies $B^{\sigma} \vdash_{\mu,\bot} A : \sigma$.

**Proof.** By simultaneous induction on $A$. We have six cases.

$A \equiv \bot$. Both (i) and (ii) are trivial by rule ($\bot$).

$A \equiv C$, where $C \in C$. Both (i) and (ii) are trivial since $C$ is well behaved.
A \equiv x. (i) We must have $x: \tau \in B^\circ$ so $\rho_{B^\circ}(x) = T^\tau$. By 4.14(i) $T^\tau \subseteq \mathcal{S}[\sigma]$ implies $\sigma \approx \tau$. We can then prove $B^\circ \vdash_{\mu, \bot} x: \sigma$ using rule ($\approx$).

(ii) Similar, using 4.14(ii).

$A \equiv x A_1 \cdots A_n$ $(n > 0)$. (i) We must have $x: \tau \in B^\circ$ and furthermore $\tau \approx T^\tau$ (recall that $\tau$ cannot be equivalent to $\mu t. t$). Moreover

$$\begin{align*}
[x A_1 \cdots A_n] \rho_{B^\circ} \\
= \{x\} \rho_{B^\circ} \cdot \{A_1\} \rho_{B^\circ} \cdot \cdots \cdot \{A_n\} \rho_{B^\circ} \\
= T^\tau \cdot \{A_1\} \rho_{B^\circ} \cdot \cdots \cdot \{A_n\} \rho_{B^\circ} \\
= \bigcup_{n \in \omega} (A^\tau \cdot n(\{A_1\} \rho_{B^\circ} \cdot T^\tau \cdot n \cup T^\tau \cdot n) \cdot \{A_2\} \rho_{B^\circ} \cdot \cdots \cdot \{A_n\} \rho_{B^\circ}) \\
= T^\tau \cdot \{A_2\} \rho_{B^\circ} \cdot \cdots \cdot \{A_n\} \rho_{B^\circ}
\end{align*}$$

assuming that $\not \not A^\tau(\{A_1\} \rho_{B^\circ})$ (use 4.11(iii)). This implies $B^\circ \vdash_{\mu, \bot} A_1: \tau$ by induction hypothesis (i). Moreover observe that

$$T^\tau \cdot \{A_2\} \rho_{B^\circ} \cdot \cdots \cdot \{A_n\} \rho_{B^\circ} = \{y A_2 \cdots A_n\} \rho_{B^\circ} \cup \{y: \tau\},$$

where $y$ is any variable not free in $B^\circ$.

We then have $\{y A_2 \cdots A_n\} \rho_{B^\circ} \cup \{y: \tau\} \subseteq \mathcal{S}[\sigma]$ and this, by induction hypothesis (i), implies $B^\circ \cup \{y: \tau\} \vdash_{\mu, \bot} (y A_2 \cdots A_n): \sigma$. Then we get a proof of $B^\circ \vdash_{\mu, \bot} (x A_1 \cdots A_n): \sigma$ by replacing the assumption $y: \tau$ with the deduction of $B^\circ \vdash_{\mu, \bot} x A_1 \cdots A_n: \tau$.

As for point (ii) we must have $\not \not A^\tau(\{y A_2 \cdots A_n\} \rho_{B^\circ} \cup \{y: \tau\})$ and we can argue as before by using induction hypothesis (ii).

Observe that the presence of the component $(\lambda v. T^\tau \cdot n)_{n+1}$ in Definition 4.10 is essential to ensure that

$$T^\tau \cdot \{A_1\} \rho_{B^\circ} \cdot \cdots \cdot \{A_n\} \rho_{B^\circ} = T^\tau \cdot \{A_2\} \rho_{B^\circ} \cdot \cdots \cdot \{A_n\} \rho_{B^\circ}.$$ 

In fact we could have $A_1 \equiv \bot$ in which case $A^\tau \cdot n(\{A_1\} \rho_{B^\circ}) \cdot T^\tau \cdot n = \{\bot\}$ for all $n$.

$A \equiv c A_1 \cdots A_n$ $(n > 0)$, where $c \in C$. This case is simple by the assumptions on $C$.

$A \equiv \lambda x. A'$. This case also is simple and is left as an exercise.

We are now able to prove the main theorem.

**Proof of Theorem 4.7 and Corollary 4.8.** Assume $B \models M: \sigma'$. Define $B'$ as the basis obtained from $B$ by replacing all occurrences of subtypes equivalent to $\mu t. t$ by $t_0$, where $t_0$ is a new type variable not occurring in $B$. Since types are ideals (i.e., downward closed), we have $B' \models A: \sigma'$ for all
$A \in \text{APP}(M)$. Now observe that for all ground substitutions $s$ we have $s(B') \models A : s(\sigma')$ (as for $t_0$ observe that $[A]^{\rho} \in \mathcal{Z}^{[\sigma'][\eta]}$ must hold for all possible choices of $I$). By 4.15 and 2.14 (which is immediately extendable to $\vdash_{\mu, \perp}$) we have $B' \vdash_{\mu, \perp} A : \sigma'$, from which we get $B \vdash_{\mu, \perp} A : \sigma$ by replacing $t_0$ with $\mu t$ (modulo $\approx$). We can then obtain $B \vdash_{\mu, C} M : \sigma$ using rule (C).

As for the corollary observe that if $M$ is in normal form we need neither rule (C) nor (C).

Since $\vdash_{\mu, C}$ on approximate normal forms is a decidable relation, finally, we have that $\vdash_{\mu, C}$ is $\Pi^0_1$. Indeed it is complete $\Pi^0_1$.

4.16. Remark. Throughout the proof of the completeness theorem we have used in an essential way the presence of basic constants in our language. The system $\vdash_{\mu, C}$ is not complete with respect to models of pure $\lambda$-calculus with approximable application (where there is no element $? \sigma$ to detect “incorrect” application) introduced in remark 3.14. In fact, in any such model $D$ the interpretation of the type $\mu t. t \rightarrow t$ is $D$ (this follows immediately from 3.10, since $D = D \rightarrow D$, where $\rightarrow$ is as in Definition 3.2(i)). This implies that $x : \sigma \models x : \mu t. t \rightarrow t$, for any type $\sigma$, but this is not formally derivable in the system $\vdash_{\mu, C}$. We don’t know which kind of rule could be introduced in order to extend the completeness result to models of the pure $\lambda$-calculus.

ACKNOWLEDGMENTS

The authors thank Professor H. Barendregt and the anonymous referees for helpful suggestions.

RECEIVED February 10, 1989; final manuscript received November 14, 1989

REFERENCES


HUET. G. (1976), "Résolution d'équations dans les langages d'ordre 1, 2, ..., ω," Thèse d'Etat, Université Paris VII.


MENDEL. N. P. (1986), First and second-order lambda calculi with recursive types, Cornell University, unpublished.


