Analysis of a guard condition in type theory (preliminary report) *

Roberto M. Amadio †
Université de Provence, Marseille

Solange Coupet-Grimal ‡
Université de Provence, Marseille

November 10, 1997

Abstract

We present a realizability interpretation of co-inductive types based on partial equivalence relations (per's). We extract from the per's interpretation sound rules to type recursive definitions. These recursive definitions are needed to introduce "infinite" and "total" objects of co-inductive type such as an infinite stream or a non-terminating process. We show that the proposed type system enjoys the basic syntactic properties of subject reduction and strong normalization with respect to a confluent rewriting system first studied by Gimenez. We also compare the proposed type system with those studied by Coquand and Gimenez. In particular, we provide a semantic reconstruction of Gimenez's system which suggests a rule to type nested recursive definitions.

1 Introduction

Coquand proposes in [Coq93] an approach to the representation of infinite objects such as streams and processes in a predicative type theory extended with *co-inductive types*. Related analyses on the role of co-inductive types (or definitions) in logical systems can be found in [Men87, LPM93] for the system F, [Pau97] for the system HOL, and [Tat94] for Beeson's Elementary theory of Operations and Numbers. Two important features of Coquand's approach are that:

• Co-inductive types, and related constructors and destructors, are added to the theory, rather than being represented by second order types and related λ-terms, as in [Geu92, Raf94].

*Appeared as *Rapport 245, Laboratoire d'Informatique de Marseille*, October 1997.

†Corresponding author. CMI (LIM), 39 rue Joliot-Curie, F-13453, Marseille, France. amadio@gyptis.univ-mrs.fr. Partially supported by CTI-CNET 95-1B-182, IFCPAR 1502-1, WG Confer, HCM Express.

‡CMI (LIM), 39 rue Joliot-Curie, F-13453, Marseille, France. solange@gyptis.univ-mrs.fr.

• Recursive definitions of infinite objects are restricted so that consideration of *partial elements* is *not* needed. Thus this work differs from work on the representation of infinite structures in lazy programming languages like Haskell (see, e.g., [Tho96]).

In his thesis, Gimenez [Gim96] has carried on a realization of Coquand's programme in the framework of the calculus of constructions [CH88]. More precisely, he studies a calculus of constructions extended with a type of streams (i.e., finite and infinite lists), and proves subject reduction and strong normalization for a related confluent rewriting system.

He also applies co-inductive types to the representation and mechanical verification of concurrent systems by relying on the Coq system [Cp96]. In this representation, processes can be *directly represented* in the logic as elements of a certain type. Hence, this approach differs sharply from those where, say, processes are represented at a syntactic level as elements of an *inductively defined* type (see, e.g., [Mel92, Nes92, AM96]). In our experience [CGJ96, CG96], the representation based on co-inductive types is more direct and manageable. This may be a decisive advantage when carrying on *formal proofs*, and it represents a solid motivation for our study.

The introduction of infinite "total" objects relies on recursive definitions which are intuitively "guarded" in a sense frequently arising in formal languages [Sal66]. An instance of the new typing rule in this approach is:

$$\frac{\Gamma, x : \sigma \vdash M : \sigma \quad M \downarrow x \quad \sigma \text{ co-inductive type}}{\Gamma \vdash \mathsf{fix}\ x.M : \sigma} . \tag{1}$$

This allows for the introduction of "infinite objects" in a "co-inductive type", by means of a "guarded" (recursive) definition. Of course, one would like to have notions of co-inductive type and of guarded definition which are as liberal as possible and that are supported by an intuitive, i.e., *semantic*, interpretation.

In Coquand's proposal, the predicate $M \downarrow x$ is defined by a straightforward analysis of the syntactic structure of the term. This is a syntactic approximation of the main issue, that is to know when the recursive definition $\mathsf{fix}\ x.M$ determines a unique total object. To answer this question we interpret co-inductive types in the category of per's (partial equivalence relations), a category of *total* computations, and we find that the guard predicate $M \downarrow x$ has a semantic analogous which can be stated as follows:

$$\forall \alpha \, ((d, e) \in \mathcal{F}_\sigma^\alpha \quad \Rightarrow \quad ([\![M]\!][d/x], [\![M]\!][e/x]) \in \mathcal{F}_\sigma^{\alpha+1}) \tag{2}$$

where $\mathcal{F}_\sigma$ is a monotonic function on per's associated to the co-inductive type σ, and $\mathcal{F}_\sigma^\alpha$ is its α^{th} iteration, for α ordinal.

We propose to represent condition (2) in the syntax by introducing some extra-notation. With the side conditions of rule (1), we introduce two types $\check{\sigma}$

and $\check{\sigma}^+$ which are interpreted respectively by $\mathcal{F}^{\alpha}_{\sigma}$ and $\mathcal{F}^{\alpha+1}_{\sigma}$. We can then replace the guard condition $M \downarrow x$ by the following typing judgment:

$$x : \check{\sigma} \vdash M : \check{\sigma}^+ \tag{3}$$

whose interpretation is basically condition (2). The revised typing system also includes:

• *Subtyping* rules which relate a co-inductive type σ to its approximations $\check{\sigma}$ and $\check{\sigma}^+$, so that we will have:

$$\sigma \leq \check{\sigma}^+ \leq \check{\sigma} \ .$$

• Rules which *overload* the constructors of the co-inductive type, e.g., if $f : \sigma \to \sigma$ is a unary constructor over σ, then f will also have the type $\check{\sigma} \to \check{\sigma}^+$ (to be understood as $\forall \alpha \ \ x \in \mathcal{F}^{\alpha}_{\sigma} \ \Rightarrow \ f(x) \in \mathcal{F}^{\alpha+1}_{\sigma}$). As it is typical of overloading, the types $\sigma \to \sigma$ and $\check{\sigma} \to \check{\sigma}^+$ which can be assigned to the constructor f turn out to be incomparable with respect to the subtyping relation.

The idea of expressing the guard condition via approximating types, subtyping, and overloading can be traced back to Gimenez's system. Our contribution here is to provide a semantic framework which:

(1) Justifies and provides an intuition for the typing rules. In particular, we will see how it is possible to understand semantically Gimenez's system.

(2) Suggests new typing rules and simplifications of existing ones. In particular, we propose: (i) a rule to type *nested recursive definitions*, and (ii) a way to type recursive definitions without labelling types.

(3) Can be readily adapted to prove *strong normalization* with respect to the confluent reduction relation introduced by Gimenez.

The following sections are organized as follows. In section 2, we introduce a simply typed λ-calculus with co-inductive types, we provide several motivating examples, and we arrive at Coquand's guard condition. In section 3, we develop an interpretation of the calculus in the category of per's over a λ-model. We analyse the problem of typing recursive definitions in the per interpretation, and we extract from this analysis a sound typing system. In section 4, we study the syntactic properties of the typing system, notably *subject reduction* and *strong normalization* with respect to a reduction relation first introduced by Gimenez. Interestingly, strong normalization can be proved by a straightforward adaptation to reducibility candidates of our per interpretation. In section 5, we present an extension of the typing system in which types are labelled. The labels can be understood as a way to avoid "interferences" among logically independent approximating types. Relying on this extension, we compare our system with Gimenez's. Finally in section 6, we point out some possible extensions which are supported by the per's interpretation.

2 A simply typed calculus

We will carry on our study in a simply typed λ-calculus extended with co-inductive types (possible extensions will be mentioned in section 6).

Types Let F be a countable set of *constructors*. We let $\mathsf{f}_1, \mathsf{f}_2, \ldots$ range over F. Let tv be the set of type variables $t, s, \ldots$ The language of *raw* types is given by the following (informal) grammar:

$$\tau ::= tv \parallel (\tau \to \tau') \parallel \nu tv.(\mathsf{f}_1 : \vec{\tau}_1 \to tv \ldots \mathsf{f}_\mathsf{k} : \vec{\tau}_k \to tv) \tag{4}$$

where $\vec{\tau}_i \to tv$ stands for $\tau_{i,1} \to \cdots \to \tau_{i,n_i} \to tv$ ($\to$ associates to the right), and all f_i are distinct. Intuitively, a type of the shape $\nu t.(\mathsf{f}_1 : \vec{\tau}_1 \to t \ldots \mathsf{f}_\mathsf{k} : \vec{\tau}_k \to t)$ is well-formed if the type variable t occurs positively in the well-formed types $\tau_{i,j}$, for $i = 1 \ldots k$, $j = 1 \ldots n_i$. Note that the type variable t is bound by ν and it can be renamed. We call types of this shape co-inductive types, the symbols $\mathsf{f}_1 \ldots \mathsf{f}_\mathsf{k}$ represent the constructors of the type. We will denote co-inductive types with the letters $\sigma, \sigma', \sigma_1, \ldots$, and unless specified otherwise, we will suppose that they have the generic form in (4). A precise definition of the well-formed types is given as follows.

Definition 1 (types) *If τ is a raw type and s is a type variable then the predicates $wf(\tau)$ (well-formed), $pos(s, \tau)$ (positive occurrence only), and $neg(s, \tau)$ (negative occurrence only) are the least predicates which satisfy the following conditions.*

(1) *If $t \in tv$ then $wf(t)$, $pos(s, t)$, and $neg(s, t)$ provided $t \neq s$.*

(2) *If $wf(\tau)$ and $wf(\tau')$ then $wf(\tau \to \tau')$. Moreover, $pos(s, \tau \to \tau')$ if $pos(s, \tau')$ and $neg(s, \tau)$, and $neg(s, \tau \to \tau')$ if $neg(s, \tau')$ and $pos(s, \tau)$.*

(3) *If $\sigma = \nu t.(\mathsf{f}_1 : \vec{\tau}_1 \to t \ldots \mathsf{f}_\mathsf{k} : \vec{\tau}_k \to t)$ and $t \neq s$ (otherwise rename t) then $wf(\sigma)$ provided $wf(\tau_{i,j})$ and $pos(t, \tau_{i,j})$ for $i = 1 \ldots k$, $j = 1 \ldots n_i$. Moreover, $pos(s, \sigma)$ if $pos(s, \tau_{i,j})$ for $i = 1 \ldots k$, $j = 1 \ldots n_i$, and $neg(s, \sigma)$ if $neg(s, \tau_{i,j})$ for $i = 1 \ldots k$, $j = 1 \ldots n_i$.*

Example 2 *Here are a few examples of well-formed co-inductive types where we suppose that the type τ is not bound by ν.*

(1) *Infinite streams over τ: $\nu s.(\mathsf{cons} : \tau \to (s \to s))$.*

(2) *Input-output processes over τ: $\nu p.(\mathsf{nil} : p, ! : \tau \to p \to p, ? : (\tau \to p) \to p)$.*

(3) *An involution: $\nu t.(\mathsf{inv} : ((t \to \tau) \to \tau) \to t)$.*

Definition 1 allows mutually recursive definitions. For instance, we can define processes over streams over processes ...:

$$\begin{aligned} \sigma &= \nu t.(\mathsf{nil} : t, ! : \sigma' \to t \to t, ? : (\sigma' \to t) \to t) \\ \sigma' &= \nu s.(\mathsf{cons} : t \to s \to s)\ . \end{aligned}$$

$$\frac{x:\tau\in\Gamma}{\Gamma\vdash x:\tau}\quad\frac{\Gamma,x:\tau\vdash M:\tau'}{\Gamma\vdash\lambda x.M:\tau\to\tau'}\quad\frac{\Gamma\vdash M:\tau'\to\tau\quad\Gamma\vdash N:\tau'}{\Gamma\vdash MN:\tau}$$

$$\text{Assuming:}\quad\begin{array}{l}\sigma=\nu t.(\mathsf{f_1}:\vec{\tau}_1\to t\ldots\mathsf{f_k}:\vec{\tau}_k\to t)\\ \vec{\tau'}\to\sigma=\tau'_1\to\ldots\tau'_m\to\sigma\quad(m\geq 0)\end{array}$$

$$\frac{}{\Gamma\vdash\mathsf{f_i}^{\sigma}:[\sigma/t]\tau_{i,1}\to\cdots[\sigma/t]\tau_{i,n_i}\to\sigma}$$

$$\frac{}{\Gamma\vdash\mathsf{case}^{\sigma}:\sigma\to([\sigma/t]\vec{\tau_1}\to\tau)\to\cdots([\sigma/t]\vec{\tau_k}\to\tau)\to\tau}$$

$$\frac{\Gamma,x:\vec{\tau'}\to\sigma\vdash M:\vec{\tau'}\to\sigma\quad M\downarrow x}{\Gamma\vdash\mathsf{fix}\ x.M:\vec{\tau'}\to\sigma}$$

Figure 1: Typing rules

These mutually recursive definitions lead to some complication in the typing of constructors. For instance, the type of cons should be $[\sigma/t](t\to\sigma'\to\sigma')$, and moreover we have to make sure that all occurrences of a cons have the same type (after unfolding). To make our analysis clearer, we prefer to gloss over these technical issues by taking a stronger definition of positivity.

Proviso 3 (restriction on positivity) *In the case (3) of definition 1 we say $pos(s,\sigma)$ (or $neg(s,\sigma)$) if s does not occur free in σ. In this way a type variable which is free in a co-inductive type cannot be bound by a ν.*

Terms Let v be the set of term variables $x,y,\ldots$ A context Γ is a possibly empty list $x_1:\tau_1\ldots x_n:\tau_n$ where all x_i are distinct. Raw terms are defined by the following grammar:

$$M ::= v \parallel (\lambda v.M) \parallel (MM) \parallel \mathsf{f}^{\sigma} \mid \mathsf{case}^{\sigma} \parallel (\mathsf{fix}\ v.M)\ . \tag{5}$$

We denote with $FV(M)$ the set of variables occurring free in the term M. The typing rules are defined in figure 1 but for the guard predicate "$M\downarrow x$". Intuitively, this predicate has to guarantee that a recursive definition does determine a unique "total" object. Before trying a formal definition, we will consider a few examples of recursive definitions, where we use the notation $\mathsf{let}\ x=M\ \mathsf{in}\ N$ for $(\lambda x.M)N$, and let application associate to the left.

Example 4 *Let o be a basic type of numerals with constants $0 : o$ and $suc : o \to o$. Let us first consider the type of infinite streams of numerals, with destructors head and tail:*

$$\sigma_1 = \nu t.(\mathsf{cons} : o \to (t \to t))$$
$$hd = \lambda x.\mathsf{case}^{\sigma_1}\ x(\lambda n.\lambda y.n) \qquad tl = \lambda x.\mathsf{case}^{\sigma_1}\ x(\lambda n.\lambda y.y)\ .$$

(1) *We can introduce an infinite list of* 0*'s as follows:*

$$\mathsf{fix}\ x.\mathsf{cons}^{\sigma_1} 0x\ .$$

(2) *We can also define a function which adds* 1 *to every element of a stream:*

$$\mathsf{fix}\ add1.\lambda x.\mathsf{case}^{\sigma_1}\ x(\lambda n.\lambda x'.\mathsf{cons}^{\sigma_1}(suc\ n)(add1\ x'))\ .$$

(3) *Certain recursive definitions should not type. For instance, consider:*

$$\mathsf{fix}\ x.\mathsf{cons}^{\sigma_1} 0(tl\ x)\ .$$

The equation does not determine a stream, as all streams of the form $\mathsf{cons}^{\sigma_1} 0z'$ *give a solution.*

(4) *The function db doubles every element in the stream:*

$$\mathsf{fix}\ db.\lambda x.\mathsf{let}\ n = (hd\ x)\ \mathsf{in}\ \mathsf{cons}^{\sigma_1} n(\mathsf{cons}^{\sigma_1} n\ db(tl\ x))\ .$$

(5) *Next we work over the type σ_2 of finite and infinite streams. The function C concatenates two streams.*

$$\sigma_2 = \nu t.(\mathsf{nil} : t, \mathsf{cons} : o \to t \to t)$$
$$C \equiv \mathsf{fix}\ conc.\lambda x.\lambda y.\mathsf{case}^{\sigma_2}\ x\ y\ \lambda n.\lambda x'.(\mathsf{cons}^{\sigma_2} n(conc\ x'\ y))\ .$$

(6) *Finally, we consider the type σ_3 of infinite binary trees whose nodes may have two colours, and the following recursive definition:*

$$\sigma_3 = \nu t.(\mathsf{bin}_1 : t \to t \to t, \mathsf{bin}_2 : t \to t \to t)$$
$$(\mathsf{fix}\ x.\mathsf{bin}_1^{\sigma_3}\ x\ (\mathsf{fix}\ y.\mathsf{bin}_2^{\sigma_3}\ x\ y))\ .$$

We recall next Coquand's definition [Coq93] of the guard predicate in the case the type theory includes just one co-inductive type, say $\sigma = \nu t.(\mathsf{nil} : t, \mathsf{cons} : o \to t \to t)$.

Definition 5 *Supposing $\Gamma, x : \vec{\tau'} \to \sigma \vdash M : \vec{\tau'} \to \sigma$, we write $M \downarrow x$ if the judgment $\Gamma, x : \vec{\tau'} \to \sigma \vdash M \downarrow_1^{\vec{\tau'} \to \sigma} x$ can be derived by the following rules, where*

n ranges over $\{0,1\}$. For the sake of readability, we omit in the premisses the conditions that $x : \vec{\tau'} \to \sigma \in \Gamma$ and the terms have the right type.

$$\frac{x \notin FV(M)}{\Gamma \vdash M \downarrow_n^{\tau} x} \qquad \frac{\Gamma, y : \tau \vdash M \downarrow_n^{\vec{\tau} \to \sigma} x \quad y \neq x}{\Gamma \vdash \lambda y.M \downarrow_n^{\tau \to \vec{\tau} \to \sigma} x}$$

$$\frac{x \notin FV(M_1) \quad \Gamma \vdash M_2 \downarrow_0^{\sigma} x}{\Gamma \vdash \mathsf{cons}^{\sigma} M_1 M_2 \downarrow_1^{\sigma} x} \qquad \frac{x \notin FV(M_1) \quad \Gamma \vdash M_2 \downarrow_0^{\sigma} x}{\Gamma \vdash \mathsf{cons}^{\sigma} M_1 M_2 \downarrow_0^{\sigma} x}$$

$$\frac{x \notin FV(N) \quad \Gamma \vdash M_1 \downarrow_n^{\sigma} x \quad \Gamma \vdash M_2 \downarrow_n^{o \to \sigma \to \sigma} x}{\Gamma \vdash \mathsf{case}^{\sigma} N M_1 M_2 \downarrow_n^{\sigma} x} \qquad \frac{x \notin FV(M_j) \quad j = 1 \dots m}{x M_1 \dots M_m \downarrow_0^{\sigma} x} \ .$$

The intuition is that "x is guarded by at least a constructor in M". Coquand's definition is quite restrictive. In particular: (i) it is unable to traverse β-redexes as in example 4(4), and (ii) it does not cope with nested recursive definitions as in example 4(6). We present in the next section a simple semantic framework which clarifies the typing issues and suggests a guard condition more powerful than the one above.

3 Interpretation

In this section we present an interpretation of the calculus in the well-known category of partial equivalence relations (per's) over a λ-model. Let $(D, \cdot, k, s, \epsilon)$ be a $\lambda\beta$-model (cf. [Bar84]). We often write de for $d \cdot e$. We denote with $A, B, \dots$ binary relations over D. We write $d\,A\,e$ for $(d,e) \in A$ and we set:

$$[d]_A = \{e \in D \mid d\,A\,e\} \quad |A| = \{d \in D \mid d\,A\,d\} \quad [A] = \{[d]_A \mid d \in |A|\} \ .$$

Definition 6 (partial equivalence relations) *Let D be a λ-model. The category of per's over D ($\mathbf{per}_D$) is defined as follows:*

$$\begin{array}{lll} \mathbf{per}_D & = \{A \mid A \subseteq D \times D \ \textit{and } A \textit{ is symmetric and transitive}\} \\ \mathbf{per}_D[A,B] & = \{f : [A] \to [B] \mid \exists\, \phi \in D \ \ (\phi \Vdash f)\} \\ \phi \Vdash f : [A] \to [B] & \textit{iff} \quad \forall\, d \in D \ \ (d \in |A| \Rightarrow \phi d \in f([d]_A)) \ . \end{array}$$

We will use the λ-notation to denote elements of the λ-model D. E.g., $\lambda x.x \Vdash f$ stands for $[\![\lambda x.x]\!]^D \Vdash f$. The category $\mathbf{per}_D$ has a rich structure, in particular it has finite products, finite sums, and exponents, whose construction is recalled in the following.

$$
\begin{gathered}
d\, A_1 \times \ldots \times A_n\, e \ \text{ iff } \ \forall\, i \in \{1 \ldots n\} \ (p_i d)\, A_i\, (p_i e) \ \text{ where:} \\
p_i = \lambda u.u(\lambda x_1 \ldots \lambda x_n.x_i) \quad p_i \| \!- \pi_i : [\Pi_{i=1 \ldots n} A_i] \to [A_i] \\
\phi_i \| \!- f_i : [C] \to [A_i] \quad \Rightarrow \quad \lambda d.\lambda u.u(\phi_1 d) \ldots (\phi_n d) \| \!- \langle f_1 \ldots f_n \rangle : [C] \to [\Pi_{i=1 \ldots n} A_i]
\end{gathered}
$$

$$
\begin{gathered}
d\, A_1 + \ldots + A_n\, e \ \text{ iff } \ \exists\, i \in \{1 \ldots n\} \ (d = (j_i d') \text{ and } (e = j_i e') \text{ and } d'\, A_i\, e') \ \text{ where:} \\
j_i = \lambda u.\lambda y_1 \ldots \lambda y_n.y_i u \quad j_i \| \!- in_i : [A_i] \to [\Sigma_{i=1 \ldots n} A_i] \\
\phi_i \| \!- f_i : [A_i] \to [C] \quad \Rightarrow \quad \lambda d.d\phi_1 \ldots \phi_n \| \!- [f_1 \ldots f_n] : [\Sigma_{i=1 \ldots n} A_i] \to [C]
\end{gathered}
$$

$$
\begin{gathered}
d\, A \to B\, e \ \text{ iff } \ \forall\, d', e' \ (d'\, A\, e' \ \Rightarrow \ (dd')\, B\, (ee')) \ \text{ where:} \\
\lambda d.(p_1 d)(p_2 d) \| \!- ev : [B^A \times A] \to B \\
\phi \| \!- f : [C \times A] \to [B] \quad \Rightarrow \quad \lambda d.\lambda d'.\phi(\lambda u.(ud)d') \| \!- \Lambda(f) : [C] \to [B^A] \ .
\end{gathered}
$$

As degenerate cases of empty product and empty sum we get terminal and initial objects:

$$
\begin{array}{ll}
1 = D \times D & \lambda x.x \| \!- f : [A] \to 1 \\
0 = \emptyset & \lambda x.x \| \!- f : [0] \to [A] \ .
\end{array}
$$

Type interpretation We denote with $\eta : tv \to \mathbf{per}_D$ type environments. The interpretation of type variables and higher types is then given as follows:

$$
[\![t]\!]_\eta = \eta(t) \qquad [\![\tau \to \tau']\!]_\eta = [\![\tau]\!]_\eta \to [\![\tau']\!]_\eta \ .
$$

As for co-inductive types, given a type $\sigma = \nu t.(\mathsf{f}_1 : \vec{\tau}_1 \to t \ldots \mathsf{f}_\mathsf{k} : \vec{\tau}_k \to t)$, and a type environment η, we define a function $\mathcal{F}_{\sigma,\eta}$ on $\mathbf{per}_D$ as follows:

$$
\mathcal{F}_{\sigma,\eta}(A) = \Sigma_{i=1 \ldots k}(\Pi_{j=1 \ldots n_i} [\![\tau_{i,j}]\!]_{\eta[A/t]}) \ . \tag{6}
$$

We then observe that $\mathbf{per}_D$ is a complete lattice with respect to set-inclusion, and that thanks to the positivity condition in the definition of co-inductive type, $\mathcal{F}_{\sigma,\eta}$ is monotonic on $\mathbf{per}_D$. Therefore we can define (*gfp* stands for greatest fixpoint):

$$
[\![\sigma]\!]_\eta = \bigcup \{A \mid A \subseteq \mathcal{F}_{\sigma,\eta}(A)\} \ \ (= gfp(\mathcal{F}_{\sigma,\eta})) \ . \tag{7}
$$

In general, if f is a monotonic function over a poset with greatest element $\top$ and glb's, we define the iteration f^α, for α ordinal as follows:

$$
f^0 = \top \quad f^{\alpha+1} = f(f^\alpha) \quad f^\lambda = \bigwedge_{\alpha < \lambda} f^\alpha \quad (\lambda \text{ limit ordinal}) \ .
$$

With this notation, we have $gfp(\mathcal{F}_{\sigma,\eta}) = \mathcal{F}^\alpha_{\sigma,\eta}$ for some ordinal α.

Example 7 *If* $\sigma = \nu t.(\mathsf{nil} : t, ! : s \to t \to t, ? : (s \to t) \to t)$ *and* $\eta(s) = B$ *then*

$$
\mathcal{F}_{\sigma,\eta}(A) = (1 + (B \times A) + (B \to A)) \ . \tag{8}
$$

Term interpretation Since $\mathbf{per}_D$ is a CCC there is a canonical interpretation of the simply typed λ-calculus. The interpretation of constructors and case is driven by equation (6). Note that to validate the typing rules it is enough to know that the interpretation of a co-inductive type is *a* fixpoint of the related functional defined by equation (6) (as a matter of fact, these rules are sound also for *inductive* types).

The interpretation of fix is more problematic. We proceed as follows:

- We define an erasure function er from the terms in the language to (pure) untyped λ-terms, and we interpret the untyped λ-terms in the λ-model D. This interpretation, is always well-defined as the λ-model accommodates arbitrary recursive definitions.
- We see what it takes for the interpretation of (the erasure of) a fixpoint to be in the corresponding type interpretation, and we derive a suitable guard condition which is expressed by additional typing rules in a suitably enriched language.
- We prove *soundness* of the interpretation with respect to the enriched typing system.

It is perhaps surprising that no explicit notion of metric is needed in our interpretation. Initially, we thought that, by analogy with the situation in formal languages [AN80], the guard condition would translate into a contraction property of an operator over a complete metric space. This suggested the use of *complete uniform per's* [Ama91]. However the simpler category of per's already provides a semantic account of the guard condition.

Definition 8 (erasure) *We define an erasure function from terms to (pure) untyped λ-terms, by induction on the structure of the term (assuming $\sigma = \nu t.(\mathsf{f_1} : \vec{\tau}_1 \to t \ldots \mathsf{f_k} : \vec{\tau}_k \to t)$).*

$$er(x) = x \quad er(\lambda x.M) = \lambda x.er(M) \quad er(MN) = er(M)er(N)$$

$$er(\mathsf{f_i}^\sigma) = \lambda x_1 \ldots \lambda x_{n_i}.\lambda y_1 \ldots \lambda y_k.y_i(\lambda u.ux_1 \ldots x_{n_i})$$

$$er(\mathsf{case}^\sigma) = \lambda x.\lambda y_1 \ldots \lambda y_k.xU(y_1) \ldots U(y_k) \quad \text{where: } U(y_i) = \lambda u.y_i(p_1 u) \ldots (p_{n_i} u)$$

$$er(\mathsf{fix}\ x.M) = Y(\lambda x.er(M)) \quad \text{where: } Y = \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))\ .$$

If $n_i = 0$ then we have $er(\mathsf{f_i}^\sigma) = \lambda y_1 \ldots \lambda y_k.y_i(\lambda u.u)$ and $U(y_i) = \lambda u.y_i$. If $k = 1$ then the definitions simplify to $er(\mathsf{f_1}^\sigma) = \lambda x_1 \ldots \lambda x_{n_1}.\lambda u.ux_1 \ldots x_{n_1}$ and $er(\mathsf{case}^\sigma) = \lambda x.\lambda y_1.y_1(p_1 x) \ldots (p_{n_1} x)$.

The erasures of $\mathsf{f_i}^\sigma$ and case^σ are designed to fit the per interpretation of co-inductive types, in particular they rely on the definition of sum and product in $\mathbf{per}_D$.

Analysis We sketch with an informal notation an instance of our semantic analysis. We write $\models P : \tau$ if $[\![P]\!]^D \in |[\![\tau]\!]|$. The typing rule for recursive definitions is sound if we can establish:

$$\models Y(\lambda x.er(M)) : \sigma \; . \tag{9}$$

Given the iterative definitions of the interpretation of the co-inductive type σ, we can try to prove:

$$\forall\, \alpha \text{ ordinal} \quad \models Y(\lambda x.er(M)) : \mathcal{F}^{\alpha}_{\sigma} \tag{10}$$

by induction on the ordinal α. The case $\alpha = 0$ is trivial since $\mathcal{F}^{\alpha}_{\sigma} = 1$, and the case α limit ordinal follows by an exchange of universal quantifications. For the case $\alpha = \alpha' + 1$, it would be enough to know:

$$\forall\, \alpha \;\; (\models Y(\lambda x.er(M)) : \mathcal{F}^{\alpha}_{\sigma} \;\; \Rightarrow \;\; \models Y(\lambda x.er(M)) : \mathcal{F}^{\alpha+1}_{\sigma}) \; . \tag{11}$$

Since $Y(\lambda x.er(M)) = [Y(\lambda x.er(M))/x]M$, property (11) is implied by the following property:

$$\forall\, \alpha, P \;\; (\models P : \mathcal{F}^{\alpha}_{\sigma} \;\; \Rightarrow \;\; \models [P/x]er(M) : \mathcal{F}^{\alpha+1}_{\sigma}) \; . \tag{12}$$

In order to represent this condition in the syntax, we parameterize the type interpretation on an ordinal α, and we introduce types $\check{\sigma}$ and $\check{\sigma}^{+}$ so that $[\![\check{\sigma}]\!]^{\alpha} = \mathcal{F}^{\alpha}_{\sigma}$, and $[\![\check{\sigma}^{+}]\!]^{\alpha} = \mathcal{F}^{\alpha+1}_{\sigma}$. Property (12) is then expressed by the judgment:

$$x : \check{\sigma} \vdash M : \check{\sigma}^{+} \; . \tag{13}$$

Revised typing rules Let T be the set of types specified in definition 1. We define the set T' as the least set such that: (i) $T \subseteq T'$, (ii) if $\sigma \in T$ is a co-inductive type then $\check{\sigma} \in T'$ and $\check{\sigma}^{+} \in T'$, and (iii) if $\tau \in T'$ and $\tau' \in T'$ then $\tau \to \tau' \in T'$. We also define the set T^{+} as the set of types in T' such that all types of the form $\check{\sigma}$ and $\check{\sigma}^{+}$ appear in positive position (the interpretation of these types is going to be anti-monotonic in the ordinal). If Γ is a context then $T(\Gamma) = \{\tau \mid x : \tau \in \Gamma\}$.

The revised typing system contains the typing rules presented in figure 1 (applied with the enriched set of types) but for the rule for fix which is replaced by the rules displayed in figure 2. Of course, all the rules are applied on the enriched set of types, and under the hypothesis that all types are well-formed.

We give some motivation and intuition for these rules. In the first rule, the condition $M \downarrow x$ is replaced by the typing judgment $\Gamma, x : \vec{\tau} \to \check{\sigma} \vdash M : \vec{\tau} \to \check{\sigma}^{+}$. The second rule for fix is used to type nested fixpoints as in example 4(6). In the rules for fix, the side conditions $T(\Gamma) \cup \{\tau'_1 \ldots \tau'_k\} \subseteq T$ and $T(\Gamma) \cup \{\tau'_1 \ldots \tau'_k\} \subseteq T^{+}$ guarantee independence and anti-monotonicity, respectively, of the type interpretation with respect to the ordinal parameter (cf. following remark 11).

$$\text{Assuming:}\quad \begin{array}{l} \sigma = \nu t.(\mathsf{f_1} : \vec{\tau}_1 \to t \ldots \mathsf{f_k} : \vec{\tau}_k \to t) \\ \vec{\tau'} \to \sigma = \tau'_1 \to \ldots \tau'_m \to \sigma \quad (m \geq 0) \end{array}$$

$$\frac{\begin{array}{c} T(\Gamma) \cup \{\tau'_1 \ldots \tau'_m\} \subseteq T \\ \Gamma, x : \vec{\tau'} \to \sigma \vdash M : \vec{\tau'} \to \sigma \\ \Gamma, x : \vec{\tau'} \to \check{\sigma} \vdash M : \vec{\tau'} \to \check{\sigma}^+ \end{array}}{\Gamma \vdash \mathsf{fix}\ x.M : \vec{\tau'} \to \sigma} \qquad \frac{\begin{array}{c} T(\Gamma) \cup \{\tau'_1 \ldots \tau'_m\} \subseteq T^+ \\ \Gamma, x : \vec{\tau'} \to \check{\sigma} \vdash M : \vec{\tau'} \to \check{\sigma}^+ \end{array}}{\Gamma \vdash \mathsf{fix}\ x.M : \vec{\tau'} \to \check{\sigma}^+}$$

$$\frac{}{\Gamma \vdash \mathsf{f_i}^\sigma : [\check{\sigma}/t]\tau_{i,1} \to \cdots \to [\check{\sigma}/t]\tau_{i,n_i} \to \check{\sigma}^+} \qquad \frac{\Gamma \vdash M : \tau \quad \tau \leq \tau'}{\Gamma \vdash M : \tau'}$$

$$\frac{}{\Gamma \vdash \mathsf{case}^\sigma : \check{\sigma}^+ \to ([\check{\sigma}/t]\vec{\tau}_1 \to \tau) \to \cdots ([\check{\sigma}/t]\vec{\tau}_k \to \tau) \to \tau}$$

$$\frac{}{\tau \leq \tau} \qquad \frac{}{\sigma \leq \check{\sigma}^+} \qquad \frac{}{\sigma \leq \check{\sigma}} \qquad \frac{}{\check{\sigma}^+ \leq \check{\sigma}} \qquad \frac{\tau_2 \leq \tau_1 \quad \tau'_1 \leq \tau'_2}{\tau_1 \to \tau'_1 \leq \tau_2 \to \tau'_2}$$

Figure 2: Additional typing rules

The additional rule for the constructors $\mathsf{f_i}$ is needed to introduce terms of type $\check{\sigma}^+$. Note that in this way we *overload* the constructors $\mathsf{f_i}$ by giving them two related types (but incomparable with respect to subtyping). The related rule which overloads the case appears to be of limited use but it is presented here to simplify the comparison with Gimenez's system in section 5.

The following rules just state the *subtyping* relations between σ, $\check{\sigma}$, and $\check{\sigma}^+$, and the way this relation is lifted higher-order. The obvious transitivity rule for the subtyping relation $\leq$ can be derived. Types with the relation $\leq$ form a quite simple partial order. In particular, if $R = \leq \cup \leq^{-1}$ then $\{\tau' \mid \tau R^* \tau'\}$ is finite.

We state some basic properties of the typing system.

Lemma 9 (1) *Exchange. If* $\Gamma, x : \tau_1, y : \tau_2, \Gamma' \vdash M : \tau$ *then* $\Gamma, y : \tau_2, x : \tau_1, \Gamma' \vdash M : \tau$ *(with a proof of the same depth).*

(2) *Remove. If* $\Gamma, x : \tau' \vdash M : \tau$ *and* $x \notin FV(M)$*, then* $\Gamma \vdash M : \tau$*.*

(3) *Weakening (restricted). If* $\Gamma \vdash M : \tau$*, x fresh, and either* $\tau' \in T$ *or* fix *does not occur in M then* $\Gamma, x : \tau' \vdash M : \tau$*.*

(4) *Transitivity. If* $\vdash \tau \leq \tau'$ *and* $\vdash \tau' \leq \tau''$ *then* $\vdash \tau \leq \tau''$*.*

(5) *Substitution. If* $\Gamma, x : \tau' \vdash M : \tau$ *and* $\Gamma \vdash N : \tau'$ *then* $\Gamma \vdash [N/x]M : \tau$*.*

Proof. (1), (2), and (3) are proved by induction on the depth of the typing proof. (4) is proved by induction on the depth of the two subtyping proofs. (5)

is proved by induction on the depth of the typing proof of M. $\square$

The terms typable using Coquand's guard condition, are strictly contained in the terms typable in the proposed typing system. This is a consequence of the following lemma.

Lemma 10 (1) *If* $\Gamma, x : \vec{\tau} \to \sigma \vdash M : \tau$, $x \notin FV(M)$, *and* M *has no occurrence of* fix, *then* $\Gamma, x : \vec{\tau} \to \check{\sigma} \vdash M : \tau$.
(2) *If* $\Gamma, x : \vec{\tau} \to \sigma \vdash M \downarrow_0^{\vec{\tau}' \to \sigma} x$ *then* $\Gamma, x : \vec{\tau} \to \check{\sigma} \vdash M : \vec{\tau}' \to \check{\sigma}$.
(3) *If* $\Gamma, x : \vec{\tau} \to \sigma \vdash M \downarrow_1^{\vec{\tau}' \to \sigma} x$ *then* $\Gamma, x : \vec{\tau} \to \check{\sigma} \vdash M : \vec{\tau}' \to \check{\sigma}^+$.

Proof. (1) is proven by lemma 9(2) (remove) and lemma 9(3) (weakening). (2) and (3) are proven by induction on the proof length. $\square$

As a matter of fact, all examples in 4 (but (3) of course) can be typed.

Revised type interpretation We parameterize the type interpretation on an ordinal α, and we define for $\sigma = \nu t.(\mathsf{f_1} : \vec{\tau}_1 \to t \ldots \mathsf{f_k} : \vec{\tau}_k \to t)$:

$$\begin{array}{llll} [\![t]\!]^\alpha_\eta & = \eta(t) & [\![\tau \to \tau']\!]^\alpha_\eta & = [\![\tau]\!]^\alpha_\eta \to [\![\tau']\!]^\alpha_\eta \\ [\![\sigma]\!]^\alpha_\eta & = gfp(\mathcal{F}_{\sigma,\eta,\alpha}) & \mathcal{F}_{\sigma,\eta,\alpha}(A) & = \Sigma_{i=1\ldots k}(\Pi_{j=1\ldots n_i}[\![\sigma]\!]^\alpha_{\eta[A/t]}) \\ [\![\check{\sigma}]\!]^\alpha_\eta & = \mathcal{F}^\alpha_{\sigma,\eta,\alpha} & [\![\check{\sigma}^+]\!]^\alpha_\eta & = \mathcal{F}^{\alpha+1}_{\sigma,\eta,\alpha} \ . \end{array}$$

Remark 11 *If* $\tau \in T$ *then* $[\![\tau]\!]^\alpha_\eta$ *does not depend on* α. *In particular, if* $\check{\sigma} \in T'$ *or* $\check{\sigma}^+ \in T'$ *then* $\sigma \in T$ *and therefore* $\mathcal{F}_{\sigma,\eta,\alpha} = \mathcal{F}_{\sigma,\eta}$. *If* $\tau \in T^+$ *and* $\alpha \leq \alpha'$ *then* $[\![\tau]\!]^\alpha_\eta \supseteq [\![\tau]\!]^{\alpha'}_\eta$, *since the types of the shape* $\check{\sigma}$ *and* $\check{\sigma}^+$ *occur in positive position.*

Let us now consider the soundness of the typing rules. If P is a pure λ-term, we write $x_1 : \tau_1 \ldots x_n : \tau_n \models P : \tau$ if

$$\forall\, \alpha, \eta \ \ ((\forall\, i \in \{1 \ldots n\} \ \ d_i \, [\![\tau_i]\!]^\alpha_\eta \, d'_i) \ \ \Rightarrow \ \ ([\![P]\!][\vec{d}/\vec{x}] \, [\![\tau]\!]^\alpha_\eta \, [\![P]\!][\vec{d'}/\vec{x}])) \ .$$

Theorem 12 (soundness) *If* $\Gamma \vdash M : \tau$ *then* $\Gamma \models er(M) : \tau$.

Proof hint. We interpret subtyping as inclusion in per's (cf. [BL90]). We write $\models \tau \leq \tau'$ if $\forall\, \eta, \alpha \ \ [\![\tau]\!]^\alpha_\eta \subseteq [\![\tau']\!]^\alpha_\eta$. First we prove that:

$$\vdash \tau \leq \tau' \ \ \Rightarrow \ \ \models \tau \leq \tau' \ .$$

Next we prove the statement by induction on the structure of the typing proof. We consider a few typical cases.

• Constructors and case. To fix the ideas suppose $\sigma = \nu t.(\mathsf{cons} : o \to t \to t)$, $O = [\![o]\!]$, and $\mathcal{F}(A) = O \times A$. Then for any per's A, B we can show:

$$\begin{array}{l} [\![er(\mathsf{cons})]\!] \in |O \to (A \to \mathcal{F}(A))| \\ [\![er(\mathsf{case})]\!] \in |\mathcal{F}(A) \to (O \to A \to B) \to B| \end{array}$$

• An instance of the argument for the first rule for fix has been outlined above, and a generalization of it is presented in the proof of proposition 21. We consider the second rule for fix. Suppose $y : \tau \vdash \mathsf{fix}\ x.M : \tau' \to \check{\sigma}^+$ is derived from $y : \tau, x : \tau' \to \check{\sigma} \vdash M : \tau' \to \check{\sigma}^+$ and $\{\tau, \tau'\} \subseteq T^+$. First we note that:

$$er(\mathsf{fix}\ x.M) = Y(\lambda x.er(M)) =_\beta [er(\mathsf{fix}\ x.M)/x]er(M) = er([\mathsf{fix}\ x.M/x]M)\ .$$

We have to show (we omit the type environment η which plays no role here):

$$\forall\, \alpha\ \ d\,[\![\tau]\!]^\alpha\, d' \ \Rightarrow\ [\![er(\mathsf{fix}\ x.M)]\!][d/y]\,[\![\tau' \to \check{\sigma}^+]\!]^\alpha\,[\![er(\mathsf{fix}\ x.M)]\!][d'/y] \tag{14}$$

knowing that:

$$\begin{array}{l}\forall \alpha\ \ d\,[\![\tau]\!]^\alpha\, d' \text{ and } e\,[\![\tau' \to \check{\sigma}]\!]^\alpha\, e' \ \Rightarrow \\ [\![er(M)]\!][d/y, e/x]\,[\![\tau' \to \check{\sigma}^+]\!]^\alpha\,[\![er(M)]\!][d'/y, e'/x]\end{array} \tag{15}$$

We proceed by transfinite induction on α. Suppose $d\,[\![\tau]\!]^\alpha\, d'$. We set $e = [\![er(\mathsf{fix}\ x.M)]\!][d/y]$ and $e' = [\![er(\mathsf{fix}\ x.M)]\!][d'/y]$. We observe: $e = [\![er(M)]\!][d/y, e/x]$ and $e' = [\![er(M)]\!][d'/y, e'/x]$.

$\alpha = 0$ By definition of $\mathcal{F}^0$ we have $e\,[\![\tau' \to \check{\sigma}]\!]^0\, e'$. By the assumption (15) we can conclude: $[\![er(M)]\!][d/y, e/x]\,[\![\tau' \to \check{\sigma}^+]\!]^0\,[\![er(M)]\!][d'/y, e'/x]$.

$\alpha = \lambda$ **limit** By the assumption $\{\tau, \tau'\} \subseteq T^+$, if $\alpha < \lambda$ then $[\![\tau]\!]^\lambda \subseteq [\![\tau]\!]^\alpha$ and $[\![\tau']\!]^\alpha \to [\![\check{\sigma}^+]\!]^\alpha \subseteq [\![\tau']\!]^\lambda \to [\![\check{\sigma}^+]\!]^\alpha$. Then we note that $\forall\, \alpha < \lambda\ e\,[\![\tau' \to \check{\sigma}^+]\!]^\alpha\, e'$ implies $\forall\, \alpha < \lambda\ \ e\,[\![\tau']\!]^\lambda \to [\![\check{\sigma}^+]\!]^\alpha\, e'$, from which we conclude $e\,[\![\tau' \to \check{\sigma}]\!]^\lambda\, e'$. We now apply the assumption (15), to derive:

$$[\![er(M)]\!][d/y, e/x]\,[\![\tau' \to \check{\sigma}]\!]^\lambda\,[\![er(M)]\!][d'/y, e'/x]\ .$$

$\alpha = \alpha' + 1$ Using again $\{\tau, \tau'\} \subseteq T^+$ we observe that $[\![\tau]\!]^\alpha \subseteq [\![\tau]\!]^{\alpha'}$ and $[\![\tau' \to \check{\sigma}^+]\!]^{\alpha'} \subseteq [\![\tau' \to \check{\sigma}^+]\!]^\alpha$. By inductive hypothesis, we know $e\,[\![\tau' \to \check{\sigma}^+]\!]^{\alpha'}\, e'$, which implies $e\,[\![\tau' \to \check{\sigma}]\!]^\alpha\, e'$. By the assumption (15), we derive: $[\![er(M)]\!][d/y, e/x]\,[\![\tau' \to \check{\sigma}^+]\!]^\alpha\,[\![er(M)]\!][d'/y, e'/x]$. $\square$

It follows from theorem 12 that: $\vdash M : \tau \ \Rightarrow\ [\![er(M)]\!] \in |[\![\tau]\!]|$. This result justifies the interpretation of a typed term as the equivalence class of its erasure (it is straightforward to adapt this interpretation to take into account contexts and environments):

$$\vdash M : \tau \ \Rightarrow\ [\![M]\!] = [[\![er(M)]\!]]_{[\![\tau]\!]}\ .$$

Next we turn to term conversion. We say that an equation $M = N : \tau$ is valid in the per interpretation, if

$$\forall\, \Gamma\ \ (\Gamma \vdash M : \tau \text{ and } \Gamma \vdash N : \tau \ \Rightarrow\ \Gamma \models M = N : \tau)\ .$$

where $x_1 : \tau_1 \dots x_n : \tau_n \models M = N : \tau$, if

$$\forall\, \alpha, \eta\ \ ((\forall\, i \in \{1 \dots n\}\ \ d_i [\![\tau_i]\!]^\alpha_\eta d'_i) \ \Rightarrow\ [\![er(M)]\!][\vec{d}/\vec{x}]\,[\![\tau]\!]^\alpha_\eta\,[\![er(N)]\!][\vec{d'}/\vec{x}])\ .$$

Reasoning at the level of erasures, it is easy to derive some valid equations.

Proposition 13 (valid equations) *The following equations are valid in the per interpretation:*

$$(\beta)\ \ (\lambda x.M)N = [N/x]M : \tau \quad (\eta)\ \ \lambda x.(Mx) = M : \tau \to \tau' \quad x \notin FV(M)$$

$$(\mathsf{case})\ \ (\mathsf{case}^\sigma\ (\mathsf{f}_\mathsf{i}^\sigma M_1 \ldots M_{n_i})\vec{N}) = N_i M_1 \ldots M_{n_i} : \tau$$

$$(\mathsf{case}_\eta)\ \ (\mathsf{case}^\sigma\ x\ \mathsf{f}_1^\sigma \ldots \mathsf{f}_\mathsf{k}^\sigma) = x : \sigma \quad (\mathsf{fix})\ \ \mathsf{fix}\ x.M = [\mathsf{fix}\ x.M/x]M : \vec{\tau} \to \sigma\ .$$

PROOF. The equations (β), (case) and (fix) follow by $\lambda\beta$-conversion of the erasures. The equations (η) and (case_η) exploit the type constraint. For (η) we prove $\lambda x.(er(M)x)\,(A \to B)\,er(M)$, by the 'extensional' definition of $(A \to B)$. For (case_η) we perform a case analysis on x. $\Box$

Proposition 14 (unique fixed point) *Suppose* $\Gamma \vdash N : \vec{\tau} \to \sigma$, $\Gamma \vdash N' : \vec{\tau} \to \sigma$, $\Gamma, x : \vec{\tau} \to \check{\sigma} \vdash M : \vec{\tau} \to \check{\sigma}^+$, $\Gamma, x : \vec{\tau} \to \sigma \vdash M : \vec{\tau} \to \sigma$, *and* $T(\Gamma) \cup \{\vec{\tau}\} \subseteq T$. *Then* $\Gamma \models [N/x]M = N : \vec{\tau} \to \sigma$ *and* $\Gamma \models [N'/x]M = N' : \vec{\tau} \to \sigma$ *implies* $\Gamma \models N = N' : \vec{\tau} \to \sigma$.

PROOF. For simplicity suppose $\Gamma \equiv y : \tau'$ and $\vec{\tau} \equiv \tau$. We prove by transfinite induction on α:

$$\forall \alpha\ \ d\,[\![\tau']\!]\,d' \ \Rightarrow\ [\![er(N)]\!][d/y]\,([\![\tau]\!] \to \mathcal{F}^\alpha)\,[\![er(N')]\!][d'/y]$$

For the case $\alpha = \alpha' + 1$ we use the guard condition on M to prove:

$$d\,[\![\tau']\!]\,d', e\,([\![\tau]\!] \to \mathcal{F}^{\alpha'})\,e' \ \Rightarrow\ [\![er(M)]\!][d/y, e/x]\,([\![\tau]\!] \to \mathcal{F}^{\alpha'})\,[\![er(M)]\!][d'/y, e'/x]$$

$$\text{where: } \begin{cases} e = [\![er(N)]\!][d/y] = [\![er(M)]\!][d/y, e/x] \text{ and} \\ e' = [\![er(N')]\!][d'/y] = [\![er(M)]\!][d'/y, e'/x]\ .\ \Box \end{cases}$$

Proposition 14 resembles Banach's theorem: contractive functions have a unique fixed point (in our case, 'contractive' is replaced by 'guarded'). Combining with unfolding (fix), one can then prove equivalences such as (cf. [Sal66]):

$$\mathsf{fix}\ x.\mathsf{cons}\ n\ (\mathsf{cons}\ n\ x) = \mathsf{fix}\ x.\mathsf{cons}\ n\ x\ .$$

We conclude this section with two definitions which 'make sense' but are not typable. Here we work with the type of infinite streams $\sigma = \nu t.(\mathsf{cons} : o \to t \to t)$:
(1) If x is a stream of numerals we denote with x_i its i^{th} element. We define a function F such that $F(x)_i = (suc^{(2^i)} x_i)$, for $i \in \omega$:

$$F \equiv \mathsf{fix}\ f.\lambda x.\mathsf{cons}^\sigma(suc(hd\ x))(f(f(tl\ x)))\ .$$

(2) A "constant" definition which determines the infinite stream of 0's.

$$\mathsf{fix}\ x.\mathsf{case}^\sigma\ x(\lambda n.\lambda y.(\mathsf{fix}\ x'.\mathsf{cons}^\sigma\ 0\ x'))\ .$$

Clearly, there is a trade-off between power and simplicity/decidability of the type system. For instance, one might consider the extension of the type system with a finite or infinite hierarchy of approximating types, say:

$$\sigma \leq \cdots \leq \check{\sigma}^{+++} \leq \check{\sigma}^{++} \leq \check{\sigma}^{+} \leq \check{\sigma} \ .$$

Our contribution here is to offer a framework in which this trade-off can be studied, and to extract from it *one possible* type system. We will see in section 4 that this "experimental" type system has some desirable syntactic properties, and in section 5 how it relates to Gimenez's system.

4 Reduction

The theory of term conversion is in general undecidable. In the presence of dependent types (like in the Calculus of Constructions), it is imperative to have an equivalence which is decidable and sufficiently powerful (otherwise type-checking becomes undecidable). A standard way to achieve decidability for an equational theory is to exhibit a rewriting system which is confluent and terminating.

Gimenez has proposed a candidate for this goal in which fix is *unfolded only under a* case (in the following we will simplify the matter by ignoring the extensional rules):

$$\begin{array}{ll}
(\lambda x.M)N & \to [N/x]M \\
\mathsf{case}^{\sigma}\ (\mathsf{f}_{\mathsf{i}}^{\sigma}\vec{M})\vec{N} & \to N_i\vec{M} \\
\mathsf{case}^{\sigma}\ ((\mathsf{fix}\ x.M)\vec{M})\vec{N} & \to \mathsf{case}^{\sigma}\ (([\mathsf{fix}\ x.M/x]M)\vec{M})\vec{N}\ .
\end{array}$$

We also denote with $\to$ the compatible closure of the rules above. It is easily seen that the resulting rewriting system is locally confluent. We verify next that the reduction rules fit well with the typing system we have proposed.

Proposition 15 (subject reduction) *If* $\Gamma \vdash M : \tau$ *and* $M \to M'$ *then* $\Gamma \vdash M' : \tau$*.*

PROOF. It is actually possible to prove this result even if we consider 'free' unfolding $(\mathsf{fix}\ x.M) \to [\mathsf{fix}\ x.M/x]M$. If $M \to M'$ then we can find a context C and a redex Δ such that $M \equiv C[\Delta] \to C[\Delta'] \equiv M'$ and $\Delta \to \Delta'$. It is easily seen that if $\Gamma \vdash M : \tau$ then $\Gamma' \vdash \Delta : \tau'$ for some Γ', τ'. If we can prove $\Gamma' \vdash \Delta' : \tau'$ then we can build a proof of $\Gamma \vdash C[\Delta'] : \tau$ by 'replacing' the proof of $\Gamma' \vdash \Delta : \tau'$ with the proof of $\Gamma' \vdash \Delta' : \tau'$.

As a typical case, let us consider the typing of $\Gamma \vdash$: $\mathsf{fix}\ x.M : \tau' \to \sigma$. This can be derived by a possibly empty sequence of subtyping rules followed by the first rule for (fix). So we derive $\Gamma \vdash \mathsf{fix}\ x.M : \tau'' \to \sigma$, where $\tau' \leq \tau''$, from $\Gamma, x : \tau'' \to \sigma \vdash M : \tau'' \to \sigma$, $\Gamma, x : \tau'' \to \check{\sigma} \vdash M : \tau'' \to \check{\sigma}^{+}$, and $T(\Gamma) \cup \{\tau''\} \subseteq T$.

Then, by the substitution lemma 9(5), we obtain $\Gamma \vdash [\mathsf{fix}\ x.M/x]M : \tau'' \to \sigma$, and by subtyping $\Gamma \vdash [\mathsf{fix}\ x.M/x]M : \tau' \to \sigma$. $\Box$

The strong normalization proof is based on an interpretation of types as reducibility candidates. Interestingly, the construction is quite similar to the one for per's. We outline the proof argument by supposing that there is just one ground type o and one co-inductive type $\sigma = \nu t.(\mathsf{cons} : o \to t \to t)$. Let SN be the set of strongly normalizing terms, when working over the language (we omit the type labels on cons and case):

$$M ::= x \mid \lambda x.M \parallel MM \parallel \mathsf{case} \parallel \mathsf{cons} \mid \mathsf{fix}\ x.M\ .$$

We say that a term is *not neutral* if it has the shape:

$$\lambda x.M,\ \mathsf{cons}\vec{M},\ (\mathsf{fix}\ x.M)\vec{M},\ \mathsf{case},\ \mathsf{case}(\mathsf{cons}M_1M_2),\ \mathsf{case}((\mathsf{fix}\ x.M)\vec{M})\ .$$

We note a fundamental property of neutral terms.

Lemma 16 *If M is neutral, then for any term N, MN and $\mathsf{case}MN$ are neutral, and they are not redexes.*

Proof. By case analysis prove that MN not neutral or MN redex implies that M is not neutral. A similar argument applies to $\mathsf{case}MN$. $\Box$

Therefore a reduction of MN (or $\mathsf{case}MN$) is either a reduction of M or a reduction of N. Following closely [GLT89], we define the collection of reducibility candidates.

Definition 17 *The set of terms X belongs to the collection RC of reducibility candidates if:*

(C_1) $X \subseteq SN$

(C_2) *If $M \in X$ and $M \to M'$ then $M' \in X$.*

(C_3) *If M is neutral and $\forall M'(M \to M' \ \Rightarrow\ M' \in X)$ then $M \in X$.*

The following are standard properties of reducibility candidates (but for (P_5) and (P_6) which *mutatis mutandis* appear in [Gim96]):

Proposition 18 *The set RC enjoys the following properties:*

(P_1) $SN \in RC$.

(P_2) *If $X \in RC$ then $x \in X$. Hence $X \neq \emptyset$.*

(P_3) *If $X, Y \in RC$ then*

$$X \to Y = \{M \mid \forall N \in X\ \ (MN \in Y)\} \in RC\ .$$

(P_4) *If* $\forall\, i \in I \;\; X_i \in RC$ *then* $\bigcap_{i \in I} X_i \in RC$.

(P_5) *If* $X \in RC$ *then*

$$\mathcal{N}(X) = \{M \mid \forall\, Y \in RC \;\; \forall\, P \in SN \to X \to Y \;\; \mathsf{case}\; MP \in Y\} \in RC \;.$$

(P_6) *If* $X \subseteq X'$ *then* $\mathcal{N}(X) \subseteq \mathcal{N}(X')$.

PROOF. We prove (P_5).
(C_1) If $M \in \mathcal{N}(X)$ take $Y = X$ and $P = \lambda n.\lambda x.x$. Then $\mathsf{case} MP \in X \subseteq SN$ implies $M \in SN$.
(C_2) Suppose $M \in \mathcal{N}(X)$ and $M \to M'$. Given $Y \in RC$ and $P \in SN \to X \to Y$, we have $\mathsf{case} MP \in Y$ and $\mathsf{case} MP \to \mathsf{case} M'P$. Hence $\mathsf{case} M'P \in Y$.
(C_3) Suppose M neutral and such that $\forall\, M' \;\; M \to M' \;\; \Rightarrow \;\; M' \in \mathcal{N}(X)$. For any Y and $P \in SN \to X \to Y$ we show $\mathsf{case} MP \in Y$ by induction on $l(P)$ where $l(P)$ is the length of the longest reduction of P. Since $\mathsf{case} MP$ is neutral, we apply (C_3) on Y. We have two cases to consider: (i) $\mathsf{case} MP \to \mathsf{case} M'P$, and (ii) $\mathsf{case} MP \to \mathsf{case} MP'$. For (i) we apply the hypothesis on M' and for (ii) we apply the inductive hypothesis on P. $\Box$

We can then define the type interpretation which is (again) parameterized on an ordinal α (of course, we take $\mathcal{N}^0 = SN$):

$$[\![o]\!]^\alpha = SN \quad [\![\tau \to \tau']\!]^\alpha = [\![\tau]\!]^\alpha \to [\![\tau']\!]^\alpha$$

$$[\![\sigma]\!]^\alpha = gfp(\mathcal{N}) \quad [\![\check{\sigma}]\!]^\alpha = \mathcal{N}^\alpha \quad [\![\check{\sigma}^+]\!]^\alpha = \mathcal{N}^{\alpha+1} \;.$$

We define $x_1 : \tau_1 \ldots x_n : \tau_n \models_{RC} M : \tau$ if $\forall\, \alpha \;\; ((\forall\, i \in \{1 \ldots n\} \;\; P_i \in [\![\tau_i]\!]^\alpha) \;\; \Rightarrow [P_1/x_1 \ldots P_n/x_n]M \in [\![\tau]\!]^\alpha)$. We can then state the following result from which strong normalization immediately follows by taking $P_i \equiv x_i$.

Proposition 19 (strong normalization) *If* $\Gamma \vdash M : \tau$ *then* $\Gamma \models_{RC} M : \tau$.

PROOF. We consider some significative cases.
$\mathsf{cons} : o \to \sigma \to \sigma$ We have to show for $Y \in RC$, $N \in SN$, $M \in gfp(\mathcal{N})$, and $P \in SN \to gfp(\mathcal{N}) \to Y$:

$$\mathsf{case}(\mathsf{cons} NM)P \in Y$$

Noting that this term is neutral, we apply (C_3) to Y and we reason by induction on the reduction lenghts.
$\mathsf{case} : \sigma \to (o \to \sigma \to \tau) \to \tau$ This follows directly by the interpretation σ as (greatest) fixed point.
fix The proof for the rules concerning fix follows the schema presented in theorem 12. An essential difference is that we cannot use unfolding here. What we use instead is condition (C_3) and a standard argument on reduction lenghts. Thus from, say, $\mathsf{case}([\mathsf{fix}\; x.[P/y]M/x]([P/y]M))R \in X$ we derive $\mathsf{case}(\mathsf{fix}\; x.[P/y]M)R \in X$.
$\Box$

Remark 20 *From these results, we can conclude that it is always better to normalize the body M of a recursive definition* $\mathsf{fix}\ x.M$*, before checking the guard condition. For instance, suppose $T(\Gamma)\cup\{\tau'_1,\ldots,\tau'_m\}\subseteq T$ and $\Gamma, x:\vec{\tau'}\to\sigma\vdash M:\vec{\tau'}\to\sigma$. From the latter and proposition 19, we know $M\to^* M'$, where M' is M's normal form. By subject reduction, we can derive $\Gamma, x:\vec{\tau'}\to\sigma\vdash M':\vec{\tau'}\to\sigma$. If we can prove $\Gamma, x:\vec{\tau'}\to\check{\sigma}\vdash M':\vec{\tau'}\to\check{\sigma}^{+}$, then we can conclude by the first typing rule for* fix*: $\Gamma\vdash\mathsf{fix}\ x.M':\vec{\tau'}\to\sigma$. On the other hand, we may fail to derive the guard condition for M and hence to type* $\mathsf{fix}\ x.M$*, e.g., consider: $M\equiv(\lambda z.\mathsf{case}\ z(\lambda n.\lambda z'.z'))(\mathsf{cons}\ n\ (\mathsf{cons}\ n\ x))$.*

5 Labelling approximated types

In the interpretation studied in section 3, all approximating types are assigned the *same* ordinal. We might consider a more liberal system in which *different* ordinals can be assigned to different approximating types. However, to express the guard condition, we still need a linguistic mechanism to say in which cases the ordinal assignment really has to be the *same*. Following this intuition, we label the approximating types with the intention to assign an ordinal to each label.

As in the previous section 4, we restrict our attention to the type of infinite streams, say σ with constructor $\mathsf{cons}: o\to\sigma\to\sigma$. The collection of types is then defined as follows:

$$\tau ::= o \parallel \sigma \parallel \sigma^{x} \parallel \sigma^{x+1} \parallel (\tau\to\tau)\ . \tag{16}$$

Roughly, we replace the type $\check{\sigma}$ with the types σ^x and the type $\check{\sigma}^{+}$ with the types σ^{x+1}, where x is a label which we take for convenience as ranging over the set of term variables $x, y, \ldots$ (any other infinite set would do). Let $var(\tau)$ be the set of variables which occur in the type τ. If Γ is a context, we also define $var(\Gamma)=\bigcup\{var(\tau)\mid x:\tau\in\Gamma\}$.

If x is a variable, we define $T^{+}(x)$ as the set of types such that all subtypes of the form σ^x or σ^{x+1} occur in positive position. The typing rules which have to be added to the system in figure 1 are shown in figure 3.

Type interpretation for labelled types Let h be an assignment from variables to ordinals. We define a type interpretation parametric in h.

$$\begin{array}{llll}
[\![o]\!]_h & = O\ \ \text{(for some chosen } per\ O) & [\![\tau\to\tau']\!]_h & = [\![\tau]\!]_h\to[\![\tau']\!]_h \\
[\![\sigma]\!]_h & = gfp(\mathcal{F}) & \mathcal{F}(A) & = O\times A \\
[\![\sigma^x]\!]_h & = \mathcal{F}^{h(x)} & [\![\sigma^{x+1}]\!]_h & = \mathcal{F}^{h(x)+1}\ .
\end{array}$$

If P is a pure λ-term, we write $x_1:\tau_1\ldots x_n:\tau_n\models P:\tau$ if

$$\forall\, h\ \ ((\forall\, i\in\{1\ldots n\}\ \ d_i\,[\![\tau_i]\!]_h\,d'_i)\ \ \Rightarrow\ \ ([\![P]\!][\vec{d}/\vec{x}]\ [\![\tau]\!]_h\,[\![P]\!][\vec{d'}/\vec{x}]))\ .$$

Assuming: $\vec{\tau'} \to \sigma^u \equiv \tau'_1 \to \cdots \to \tau'_m \to \sigma^u \quad (m \geq 0, u \text{ can be a label or nothing})$

$$\frac{x \notin var(\Gamma) \cup \bigcup\{var(\tau'_i) \mid i = 1 \ldots m\} \quad \Gamma, x : \vec{\tau'} \to \sigma^x \vdash M : \vec{\tau'} \to \sigma^{x+1} \quad \Gamma, x : \vec{\tau'} \to \sigma \vdash M : \vec{\tau'} \to \sigma}{\Gamma \vdash \mathsf{fix}\ x.M : \vec{\tau'} \to \sigma}$$

$$\frac{T(\Gamma) \cup \{\tau'_i \mid i = 1 \ldots m\} \subseteq T^+(y) \quad \Gamma, x : \vec{\tau'} \to \sigma^y \vdash M : \vec{\tau'} \to \sigma^{y+1}}{\Gamma \vdash \mathsf{fix}\ x.M : \vec{\tau'} \to \sigma^{y+1}}$$

$$\frac{}{\Gamma \vdash \mathsf{cons} : o \to \sigma^x \to \sigma^{x+1}} \qquad \frac{\Gamma \vdash M : \tau \quad \tau \leq \tau'}{\Gamma \vdash M : \tau'}$$

$$\frac{}{\Gamma \vdash \mathsf{case} : \sigma^{x+1} \to (o \to \sigma^x \to \tau) \to \tau}$$

$$\frac{}{\tau \leq \tau} \qquad \frac{}{\sigma \leq \sigma^{x+1}} \qquad \frac{}{\sigma \leq \sigma^x} \qquad \frac{}{\sigma^{x+1} \leq \sigma^x} \qquad \frac{\tau_2 \leq \tau_1 \quad \tau'_1 \leq \tau'_2}{\tau_1 \to \tau'_1 \leq \tau_2 \to \tau'_2}$$

Figure 3: Additional typing rules for labelled types

We revisit the soundness proof (cf. theorem 12) using the typing rules in figure 3.

Proposition 21 *If* $\Gamma \vdash M : \tau$ *then* $\Gamma \models er(M) : \tau$.

Proof hint. We write $\models \tau \leq \tau'$ if $\forall h \ [\![\tau]\!]_h \subseteq [\![\tau']\!]_h$. First we prove that:

$$\vdash \tau \leq \tau' \quad \Rightarrow \quad \models \tau \leq \tau' \ .$$

Next we prove the statement by induction on the structure of the typing proof. We consider the soundness for the two rules concerning fix. For the sake of readability, we will consider a simplified judgment of the form:

$$y : \tau \vdash \mathsf{fix}\ x.M : \tau' \to \sigma^u \ .$$

In order to make the basic argument clearer, we will work with an informal notation. Thus we will use 'type-assignment' rather than 'relational equivalence', and we will write $M \in \tau_h$ rather than $[\![er(M)]\!] \in |[\![\tau]\!]_h|$. For the first rule we have to prove:

$$\forall h \ \ y \in \tau_h \quad \Rightarrow \quad \mathsf{fix}\ x.M \in (\tau' \to \sigma)_h \tag{17}$$

knowing that:

$$\forall h \ \ y \in \tau_h, x \in (\tau' \to \sigma^x)_h \quad \Rightarrow \quad M \in (\tau' \to \sigma^{x+1})_h \ . \tag{18}$$

The statement (17) can be rephrased as:

$$\forall\, \alpha, h \;\; y \in \tau_h \quad \Rightarrow \quad \mathsf{fix}\ x.M \in (\tau' \to \sigma^x)_{h[\alpha/x]}\ . \tag{19}$$

This follows by the interpretation of σ, and the side-conditions that guarantee that the other types do not depend on x. We show (19) by transfinite induction on α.

$\alpha = 0$ Then every term is in $(\tau' \to \sigma^x)_{h[0/x]}$, by definition of $\mathcal{F}^0$.

$\alpha = \lambda$ Suppose $y \in \tau_h$. By inductive hypothesis, $\forall\, \alpha < \lambda$ $\mathsf{fix}\ x.M \in (\tau' \to \sigma^x)_{h[\alpha/x]}$. Since $\sigma^x_{h[\lambda/x]} = \bigcap_{\alpha<\lambda} \sigma^x_{h[\alpha/x]}$, we can conclude $\mathsf{fix}\ x.M \in (\tau' \to \sigma^x)_{h[\lambda/x]}$.

$\alpha = \alpha' + 1$ Suppose $y \in \tau_h$. By inductive hypothesis $\mathsf{fix}\ x.M \in (\tau' \to \sigma^x)_{h[\alpha/x]}$. Now we apply the assumption (18) with $h' = h[\alpha/y]$ to derive:

$$y \in \tau_{h'}, x \in (\tau' \to \sigma^x)_{h'} \quad \Rightarrow \quad M \in (\tau' \to \sigma^{x+1})_{h'}\ .$$

From this we conclude:

$$[\mathsf{fix}\ x.M/x]M = \mathsf{fix}\ x.M \in (\tau' \to \sigma^{x+1})_{h[\alpha/x]}\ .$$

We now turn to the second rule for fix. We have to show:

$$\forall\, h \;\; y \in \tau_h \quad \Rightarrow \quad \mathsf{fix}\ x.M \in (\tau' \to \sigma^{y+1})_h \tag{20}$$

knowing

$$\forall\, h \;\; y \in \tau_h, x \in (\tau' \to \sigma^y)_h \quad \Rightarrow \quad M \in (\tau' \to \sigma^{y+1})_h\ . \tag{21}$$

We reformulate (20) as

$$\forall\, \alpha, h \;\; y \in \tau_{h[\alpha/y]} \quad \Rightarrow \quad \mathsf{fix}\ x.M \in (\tau' \to \sigma^{y+1})_{h[\alpha/y]} \tag{22}$$

and we prove it by transfinite induction on α.

$\alpha = 0$ Suppose $y \in \tau_{h[0/y]}$. By definition of $\mathcal{F}^0$, we know $\mathsf{fix}\ x.M \in (\tau' \to \sigma^y)_{h[0/y]}$. By (21), we derive $[\mathsf{fix}\ x.M/x]M = \mathsf{fix}\ x.M \in (\tau' \to \sigma^{y+1})_{h[0/y]}$.

$\alpha = \lambda$ Suppose $y \in \tau_{h[\lambda/y]}$. By the positivity condition, $y \in \tau_{h[\alpha/y]}$, for $\alpha < \lambda$. By inductive hypothesis, we derive $\forall\, \alpha < \lambda$ $\mathsf{fix}\ x.M \in (\tau' \to \sigma^{y+1})_{h[\alpha/y]}$. Suppose $z \in \tau'_{h[\lambda/y]}$, again by the positivity condition $z \in \tau'_{h[\alpha/y]}$, for $\alpha < \lambda$. But this implies, by definition of $\mathcal{F}^\lambda$, $\forall\, \alpha < \lambda$ $\mathsf{fix}\ x.M \in (\tau' \to \sigma^y)_{h[\lambda/y]}$. Now we use (21) to conclude: $[\mathsf{fix}\ x.M/x]M = \mathsf{fix}\ x.M \in (\tau' \to \sigma^{y+1})_{h[\lambda/y]}$.

$\alpha = \alpha' + 1$ Suppose $y \in \tau_{h[\alpha'+1/y]}$. By the positivity condition, $y \in \tau_{h[\alpha'/y]}$. By inductive hypothesis, $\mathsf{fix}\ x.M \in (\tau' \to \sigma^{y+1})_{h[\alpha'/y]}$. By positivity of τ', $\mathsf{fix}\ x.M \in (\tau' \to \sigma^y)_{h[\alpha/y]}$. Now we apply (21) to get: $[\mathsf{fix}\ x.M/x]M = \mathsf{fix}\ x.M \in (\tau' \to \sigma^{y+1})_{h[\alpha/y]}$. $\square$

Comparison with Gimenez's system In his thesis, Gimenez has studied an extension of the calculus of constructions with the co-inductive type of finite and infinite streams (cf. example 4(5)).

In the Coq system, the user can actually introduce other co-inductive types. Among the examples of co-inductive type considered in this paper, the type in example 2(3) is the only one which is rejected. The reason is that Coq relies on a stricter notion of positivity to avoid some consistency problems which arise at higher-order types [Gim97]. It should be noted that Coq implementation of co-inductive types was developed *before* the type theory was settled, and cannot be considered as a faithful implementation of it.

When Gimenez's system is considered in a simply-typed framework, the following differences appear with respect to the system presented in figure 3 (ignoring some minor notational conventions):

(1) Gimenez's typing system is presented in a "Church" style. More precisely, the variables bound by λ and fix carry a type, and this type is used to constraint (in the usual way) the application of the related typing rules.

(2) The subtyping rule for functional types $\tau \to \tau'$ is missing.

(3) The second rule for typing recursive definitions is missing.

Obviously these differences imply that one can give *less* types to a term in Gimenez's system than in our system. To be fair, one has to notice that the presentation as a Church system and the absence of subtyping at higher-types is essentially justified by the complexity of the calculus of constructions, and by the desire to avoid too many complications at once. On the other hand, the lack of the second rule for fix is, in our opinion, a genuine difference, which moreover has an impact in practice, as the rule is needed to type nested recursive definitions as that of example 4(6).

A question which should be raised is whether the system with type labels in figure 3 is better *in practice* than the simpler system without type labels in figure 2. So far, we could not find any "natural" example suggesting a positive answer.

6 Conclusion

We have proposed an interpretation of co-inductive types in per-models, and derived from this interpretation sound rules to type recursive definitions of objects of co-inductive type. Per's interpretations support other relevant extensions of the type theory, including second-order types (see, e.g., [LM92]) and inductive types (see, e.g., [Loa97]). As expected, an inductive type, e.g., $\mu t.(\mathsf{nil} : t, \mathsf{cons} : o \to t \to t)$ is interpreted as the *least fixpoint* of the operator $\mathcal{F}$ described in section 3. It follows that there is a natural subtyping relation between the inductive type and the corresponding co-inductive type $\nu t.(\mathsf{nil} : t, \mathsf{cons} : o \to t \to t)$. The study of a type theory which combines these features is an attractive topic for further investigation. Other important issues include: (i) an analysis of the *complexity* the type reconstruction problem, and (ii) an investigation of the *completeness* of the type system with respect to a modified per interpretation based on a term model construction à la Henkin.

Acknowledgement Section 5 relies on an e-mail exchange with Eduardo Gimenez who provided the simply typed formulation of his system and explained the motivations behind certain design choices. His help is gratefully acknowledged.

References

[AM96] O. Ait-Mohamed. *La théorie du π-calcul dans le système HOL.* PhD thesis, Université de Nancy, 1996.

[Ama91] R. Amadio. Recursion over realizability structures. *Information and Computation*, 91(1):55–85, 1991.

[AN80] A. Arnold and M. Nivat. Metric interpretation of infinite trees and semantics of non-deterministic recursive programs. *Theoretical Computer Science*, 11:181–205, 1980.

[Bar84] H. Barendregt. *The lambda calculus; its syntax and semantics.* North-Holland, 1984.

[BL90] K. Bruce and G. Longo. A modest model of records, inheritance and bounded quantification. *Information and Computation*, 87:196–240, 1990.

[CG96] S. Coupet-Grimal. Observational equivalence relations for transition systems (Coq sources). Distributed with the Coq system, 1996.

[CGJ96] S. Coupet-Grimal and L. Jakubiec. Coq and hardware verification: a case study. In *Proc. Theorem Proving in higher order logics, Springer Lect. Notes in Comp. Sci. 1125*, 1996.

[CH88] T. Coquand and G. Huet. A calculus of constructions. *Information and Computation*, 76:95–120, 1988.

[Coq93] T. Coquand. Infinite objects in type theory. In *Types for proofs and programs, Springer Lect. Notes in Comp. Sci. 806*, 1993.

[Cp96] Coq-project. The Coq proof assistant reference manual. http://pauillac.inria.fr/coq, 1996.

[Geu92] H. Geuvers. Inductive and coinductive types with iteration and recursion. In *Proc. of Workshop on types for proofs and programs, Nordström et al. (eds.)*, pages 193–217, 1992. Available electronically.

[Gim96] E. Gimenez. *Un calcul de constructions infinies et son application à la vérification de systèmes communicants.* PhD thesis, ENS Lyon, 1996.

[Gim97] E. Gimenez. Personal communication. October 1997.

[GLT89] J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types.* Cambridge University Press, 1989.

[LM92] G. Longo and E. Moggi. Constructive natural deduction and its modest interpretation. *Mathematical Structures in Computer Science*, 1:215–254, 1992.

[Loa97] R. Loader. Equational theories for inductive types. *Annals of Pure and Applied Logic*, 1997. To appear.

[LPM93] F. Leclerc and C. Paulin-Morhing. Programming with streams in Coq. A case study: the sieve of Eratosthenes. In *Proc. TYPES, Springer Lect. Notes in Comp. Sci. 806*, 1993.

[Mel92] T. Melham. A mechanized theory of π-calculus in HOL. Technical Report 244, Computer Laboratory, University of Cambridge, January 1992.

[Men87] N. Mendler. Recursive types and type constraints in second-order lambda calculus. In *Proc. IEEE Logic in Comp. Sci.*, 1987.

[Nes92] M. Nesi. A formalization of the process algebra CCS in higher order logic. Technical Report 278, Computer Laboratory, University of Cambridge, December 1992.

[Pau97] L. Paulson. Mechanizing coinduction and corecursion in higher-order logic. *J. of Logic and Computation*, 7(2):175–204, 1997.

[Raf94] C. Raffalli. *L'arithmétique fonctionnelle du second ordre avec point fixes.* PhD thesis, Université Paris VII, 1994.

[Sal66] A. Salomaa. Two complete systems for the algebra of complete events. *Journal of the ACM*, 13-1, 1966.

[Tat94] M. Tatsuta. Realizability interpretation of coinductive definitions and program synthesis with streams. *Theoretical Computer Science*, 122:119–136, 1994.

[Tho96] S. Thompson. *Haskell. The craft of functional programming.* Addison-Wesley, 1996.