Theoretical Informatics and Applications Will be set by the publisher
Informatique Théorique et Applications

MONOTONE (CO)INDUCTIVE TYPES AND POSITIVE FIXED-POINT TYPES *

Ralph Matthes[1]

Abstract. We study five extensions of the polymorphically typed lambda-calculus (system F) by type constructs intended to model fixed-points of monotone operators. Building on work by H. Geuvers concerning the relation between term rewrite systems for least pre-fixed-points and greatest post-fixed-points of positive type schemes (i. e., non-nested positive inductive and coinductive types) and so-called retract types, we show that there are reduction-preserving embeddings even between systems of monotone (co)inductive types and non-interleaving positive fixed-point types (which are essentially those retract types). The reduction relation considered is β- and η-reduction for system F plus either (full) primitive recursion on the inductive types or (full) primitive corecursion on the coinductive types or an extremely simple rule for the fixed-point types. Monotonicity is not confined to the syntactic restriction on type formation of having only positive occurrences of the type variable α in ρ for the inductive type $\mu\alpha\rho$ or the coinductive type $\nu\alpha\rho$. Instead of that only a "monotonicity witness" which is a term of type $\forall\alpha\forall\beta.(\alpha \to \beta) \to \rho \to \rho[\alpha := \beta]$ is required. This term may already use (co)recursion such that our monotone (co)inductive types may even be "interleaved" and not only nested.

AMS Subject Classification. 03B40, 68Q42, 68Q65.

Keywords and phrases: system F, monotonicity witness, monotone inductive type, monotone coinductive type, retract type, primitive recursion, primitive corecursion, iteration, coiteration

* *I am very thankful for support by the Volkswagenstiftung.*

[1] LFE für theoretische Informatik, Institut für Informatik der Universität München, Oettingenstraße 67, D-80538 München, Germany; email: matthes@informatik.uni-muenchen.de

1. Introduction

Our goal is to establish relations between extensions of system F (the polymorphic λ-calculus due to Girard [4] and Reynolds [10]) with

- inductive types,
- coinductive types and
- fixed-point types.

Why do we need to extend system F by these type constructs? The main problem is that not (full) primitive recursion but only iteration on inductive types is modelled by the impredicative encoding of inductive types in system F [5, 8]. Dually, only coiteration on coinductive types but not (full) primitive corecursion is achieved by the impredicative encoding of coinductive types.

In this paper, a "relation" shall mean a type-respecting reduction-preserving embedding.

Definition 1.1. A type-respecting reduction-preserving embedding (embedding for short) of a typed term rewrite system $\mathcal{S}$ into a typed term rewrite system $\mathcal{S}'$ is a function $-'$ (the $-$ sign represents the indefinite argument of the function $'$) which assigns to every type ρ of $\mathcal{S}$ a type ρ' of $\mathcal{S}'$ and to every term r of type ρ of $\mathcal{S}$ a term r' of the (image) type ρ' of $\mathcal{S}'$ such that the following implication holds: If $r \to s$ in $\mathcal{S}$, then $r' \to^+ s'$ in $\mathcal{S}'$. ($\to^+$ denotes the transitive closure of $\to$.)

In short, an embedding $-'$ is a pair of functions both denoted by $-'$ which are compatible, and such that through $-'$ one rewrite step in the source system is simulated by at least one step in the target system. Obviously we have that a system which embeds into a strongly normalizing system is itself strongly normalizing, i. e., has no infinite reduction sequences. Setting up embeddings into strongly normalizing systems is thus an efficient way of proving strong normalization for the proposed extensions of system F which are

- the system NPI of non-interleaving positive inductive types,
- the system NPC of non-interleaving positive coinductive types,
- the system MI of monotone inductive types,
- the system MC of monotone coinductive types and
- the system NPF of non-interleaving positive fixed-point types.

In fact, we prove that all of them embed into each other which shows that w. r. t. our notion of embedding, the above-mentioned defect of system F is overcome by adding a subset of all possible positive fixed-point types and even arrive at full primitive recursion and corecursion for any monotone (co)inductive type. Moreover, strong normalization for all the systems follows from strong normalization of any of them. In [8] a direct proof of strong normalization is given for NPF, and in [7] a direct proof for MI. It is an exercise to extend M. Takahashi's confluence proof [11] to these systems (see [7] for β-reduction in NPI; confluence is easy to establish because of the absence of nontrivial critical pairs in all our systems; however, confluence is not inherited via embeddings). Hence, the equality theory of all our extensions is decidable.

2. The base system and its extensions

We introduce a version of system F and several extensions. They are typed λ-calculi where every term has its type and the presentation of the term rules strictly follows the idea of natural deduction proof systems.

2.1. System F

The version of system F we will use is in essence the same as in [5].

Types: We have infinitely many type variables (denoted by α, β, ...) and with types ρ and σ we also have the product type $\rho \times \sigma$ and the function type $\rho \to \sigma$. Moreover, given a variable α and a type ρ we form the universal type $\forall\alpha\rho$. The quantifier $\forall$ binds α in ρ. The *renaming convention* for bound variables is adopted, i. e., we syntactically identify types which only differ in the names of their bound type variables. The definition of the result $\rho[\alpha := \sigma]$ of the *substitution* of σ for α in ρ is then easily defined. Let $\mathsf{FV}(\rho)$ be the set of type variables occurring free in ρ.

The *terms* of F are presented without contexts and with fixed types (see [2] p. 159 for comments on this original typing à la Church). We have infinitely many term variables with types (denoted e. g. by x^ρ), pairing $\langle r^\rho, s^\sigma\rangle^{\rho\times\sigma}$, projections $(r^{\rho\times\sigma}\mathsf{L})^\rho$ and $(r^{\rho\times\sigma}\mathsf{R})^\sigma$, λ-abstraction $(\lambda x^\rho r^\sigma)^{\rho\to\sigma}$ for terms, term application $(r^{\rho\to\sigma}s^\rho)^\sigma$, λ-abstraction $(\Lambda\alpha r^\rho)^{\forall\alpha\rho}$ for types (under the usual proviso that α does not occur free in the type of any variable free in r) and type application $(r^{\forall\alpha\rho}\sigma)^{\rho[\alpha:=\sigma]}$. We also write $r : \rho$ for "the term r has type ρ". We freely use the (analogous) *renaming convention* for bound term and type variables of terms, e. g. for defining the substitution of types for type variables in terms—written $r[\alpha := \sigma]$—and the substitution of terms for term variables of the same type in terms—written $r[x^\rho := s]$—appropriately.

It turns out that a term variable in fact has to be defined as a pair consisting of a variable name and a type. Hence, a slight ambiguity arises with the standard practice of omitting type superscripts because the x in x^ρ is only an untyped variable name. Nevertheless we follow the standard practice. The interested reader may consult the discussion in [7], sections 2.1.2 and 2.2.6. (In 2.1.2 the intricate problem of dealing with terms having free variables of different types with the same variable name is studied in great detail.) Let $\mathsf{FTV}(r)$ be the set of free type variables in r and $\mathsf{FV}(r)$ be the set of term variables occurring free in r.

Definition 2.1. Beta plus eta reduction $\mapsto$ for system F is as usual given by

$$
\begin{array}{cl}
(\beta_\times) & \langle r, s\rangle\mathsf{L} \mapsto r \\
 & \langle r, s\rangle\mathsf{R} \mapsto s \\
(\eta_\times) & \langle r\mathsf{L}, r\mathsf{R}\rangle \mapsto r \\
(\beta_\to) & (\lambda x^\rho r)s \mapsto r[x^\rho := s] \\
(\eta_\to) & \lambda x^\rho.rx \mapsto r \quad \text{if } x \text{ is not free in } r \\
(\beta_\forall) & (\Lambda\alpha r)\sigma \mapsto r[\alpha := \sigma] \\
(\eta_\forall) & \Lambda\alpha.r\alpha \mapsto r \quad \text{if } \alpha \text{ is not free in } r.
\end{array}
$$

The reduction relation $\to$ is defined as the term closure of $\mapsto$. Equivalently, $r[x := s] \to r[x := s']$ iff $s \mapsto s'$ and r has exactly one free occurrence of x. We denote the reflexive transitive closure of $\to$ by $\to^*$. It is well-known that F *has subject reduction*, i. e., if $r : \rho$ and $r \to r'$, then $r' : \rho$. This will also be the case for all of the systems to be defined in the sequel. Strong normalization of F is a famous result by Girard [4] (eta reduction requires but a modification of the proof for beta only or, alternatively, the result follows from strong beta normalization by eta postponement). As mentioned in the introduction, confluence is easily established by Takahashi's method [11]. The type former $\to$ is assumed to associate to the right and application to the left which fits well: $\rho \to \sigma \to \tau := \rho \to (\sigma \to \tau)$ and $r^{\rho\to\sigma\to\tau} s^\rho t^\sigma := (rs)t$.

2.2. Non-interleaving positive inductive types

The system which will be called NPI moreover has inductive types $\mu\alpha\rho$ for α only occurring positively (not necessarily strictly positively—i. e., we define negative occurrences in parallel with positive occurrences) in ρ and not free in some subexpression $\mu\alpha'\rho'$ of $\mu\alpha\rho$.

We now give a precise definition of the type system.

Definition 2.2. Inductively define the set NPTy of non-interleaved positive types and simultaneously for every $\rho \in \mathsf{NPTy}$ the set $\mathsf{NPos}(\rho)$ of type variables which only occur free at positive positions in ρ and are not in the scope of an application of the μ-rule and the set $\mathsf{NNeg}(\rho)$ of type variables which only occur free at negative positions in ρ and are not in the scope of an application of the μ-rule as follows:

- (V) $\alpha \in \mathsf{NPTy}$ and $\mathsf{NPos}(\alpha) :=$ all type variables, and $\mathsf{NNeg}(\alpha) :=$ all type variables except α.
- ($\times$) If $\rho \in \mathsf{NPTy}$ and $\sigma \in \mathsf{NPTy}$, then $\rho \times \sigma \in \mathsf{NPTy}$.
  $\mathsf{NPos}(\rho \times \sigma) := \mathsf{NPos}(\rho) \cap \mathsf{NPos}(\sigma)$.
  $\mathsf{NNeg}(\rho \times \sigma) := \mathsf{NNeg}(\rho) \cap \mathsf{NNeg}(\sigma)$.
- ($\to$) If $\rho \in \mathsf{NPTy}$ and $\sigma \in \mathsf{NPTy}$, then $\rho \to \sigma \in \mathsf{NPTy}$.
  $\mathsf{NPos}(\rho \to \sigma) := \mathsf{NNeg}(\rho) \cap \mathsf{NPos}(\sigma)$.
  $\mathsf{NNeg}(\rho \to \sigma) := \mathsf{NPos}(\rho) \cap \mathsf{NNeg}(\sigma)$.
- ($\forall$) If $\rho \in \mathsf{NPTy}$, then $\forall\alpha\rho \in \mathsf{NPTy}$.
  $\mathsf{NPos}(\forall\alpha\rho) := \mathsf{NPos}(\rho) \cup \{\alpha\}$ and $\mathsf{NNeg}(\forall\alpha\rho) := \mathsf{NNeg}(\rho) \cup \{\alpha\}$.
- (μ) If $\rho \in \mathsf{NPTy}$ and $\alpha \in \mathsf{NPos}(\rho)$, then $\mu\alpha\rho \in \mathsf{NPTy}$.
  $\mathsf{NPos}(\mu\alpha\rho) := \mathsf{NNeg}(\mu\alpha\rho) :=$ all type variables except those in $\mathsf{FV}(\mu\alpha\rho)$.

It is not hard to show that $\alpha \in \mathsf{NPos}(\rho) \cap \mathsf{NNeg}(\rho)$ whenever $\alpha \notin \mathsf{FV}(\rho)$ and that type substitution (with terms from NPTy) does not lead out of the set NPTy.

Intuitively, $\mu\alpha\rho$ is the least fixed-point of the mapping $\sigma \mapsto \rho[\alpha := \sigma]$.

Examples 2.3. Assume the canonical impredicative encodings $1 := \forall\alpha.\alpha \to \alpha$ and $\rho + \sigma := \forall\beta.(\rho \to \beta) \to (\sigma \to \beta) \to \beta$ (for $\beta \notin \mathsf{FV}(\rho) \cup \mathsf{FV}(\sigma)$; ρ and σ are at non-strict positive positions in $\rho + \sigma$ which amounts to saying that $\alpha', \alpha \in \mathsf{NPos}(\alpha' + \alpha)$; clearly, $\beta \notin \mathsf{NPos}((\rho \to \beta) \to (\sigma \to \beta) \to \beta))$. Then the natural

numbers are modelled by
$$\mathsf{nat} := \mu\alpha.1 + \alpha$$
and the ρ-branching well-founded trees have the type (with $\beta \notin \mathsf{FV}(\rho)$)
$$\mathsf{tree}(\rho) := \mu\beta.1 + (\rho \to \beta).$$
Hence, $\mathsf{tree}(\mathsf{nat})$ is the nested inductive type of "ordinals". We rule out interleaving of μ: Writing $\mathsf{list}(\rho) := \mu\beta.1 + \rho \times \beta$ we do not have the type $\mu\alpha.\mathsf{list}(\alpha)$ in our type system (although the dependency is only strictly positive): $\mathsf{list}(\alpha) \in \mathsf{NPTy}$ (every type variable is in $\mathsf{NPos}(1 + \alpha \times \beta)$), but $\alpha \notin \mathsf{NPos}(\mathsf{list}(\alpha))$ = all type variables except α.

For the same reason we do not have the type $\mu\alpha.1 + (\mathsf{tree}(\alpha) \to \alpha)$ which is non-strictly positive and interleaving.

Obviously, also $\mu\alpha.\alpha \to \alpha$ does not fit into the type system ($\mathsf{NPos}(\alpha \to \alpha)$ = all type variables except α).

How do we introduce (full) primitive recursion? Given a type $\mu\alpha\rho \in \mathsf{NPTy}$ and a term s of type $\rho[\alpha := \mu\alpha\rho \times \sigma] \to \sigma$, we follow [3] (taking its motivation from initial algebras of functors in category theory) and postulate the existence of a term $\mathrm{Rec}_{\mu\alpha\rho} s$ such that the diagram in figure 1 commutes. (Explanation: $\mathrm{C}_{\mu\alpha\rho}$ "folds" $\rho[\alpha := \mu\alpha\rho]$ into $\mu\alpha\rho$ and establishes one direction of the intuitive isomorphism $\rho[\alpha := \mu\alpha\rho] \cong \mu\alpha\rho$, Id denotes the identity $\lambda x^{\mu\alpha\rho} x$ on $\mu\alpha\rho$, $\langle\cdot,\cdot\rangle$ is pairing defined pointwise and $\rho[\cdot]$ shall mean the canonical lifting of $\lambda\alpha\rho$ to terms discussed below. Composition is taken from simply-typed lambda-calculus.)

$$\begin{tikzcd}
\rho[\alpha := \mu\alpha\rho] \arrow[r, "\mathrm{C}_{\mu\alpha\rho}"] \arrow[d, "{\rho[\langle \mathrm{Id}, \mathrm{Rec}_{\mu\alpha\rho} s\rangle]}"'] & \mu\alpha\rho \arrow[d, "\mathrm{Rec}_{\mu\alpha\rho} s"] \\
\rho[\alpha := \mu\alpha\rho \times \sigma] \arrow[r, "s"'] & \sigma
\end{tikzcd}$$

FIGURE 1. (full) primitive recursion

In this preliminary version we would assume a constant $\mathrm{C}_{\mu\alpha\rho}$ having the type $\rho[\alpha := \mu\alpha\rho] \to \mu\alpha\rho$ and a term former $\mathrm{Rec}_{\mu\alpha\rho}$ such that for terms s of type $\rho[\alpha := \mu\alpha\rho \times \sigma] \to \sigma$, $\mathrm{Rec}_{\mu\alpha\rho} s$ is a term of type $\mu\alpha\rho \to \sigma$, and introduce the equality axiom
$$(\mathrm{Rec}_{\mu\alpha\rho} s) \circ \mathrm{C}_{\mu\alpha\rho} = s \circ \rho[\langle \mathrm{Id}, \mathrm{Rec}_{\mu\alpha\rho} s\rangle].$$
(We do not require $\mathrm{Rec}_{\mu\alpha\rho} s$ to be unique with this property since we study intensional equality.)

However, since our goal is the study of rewrite systems we have to direct this equation. Moreover, some care has to be taken in order to get embeddings instead of only equation-preserving translations. We therefore strictly adhere to a natural deduction formulation. This directly leads to the following *term formation rules* for NPI:

(μ-I) If t is a term of type $\rho[\alpha := \mu\alpha\rho]$, then $\mathrm{C}_{\mu\alpha\rho}t$ is a term of type $\mu\alpha\rho$.

(μ-E) If r is a term of type $\mu\alpha\rho$ and s is a term of type $\rho[\alpha := \mu\alpha\rho \times \sigma] \to \sigma$, then $r\mathrm{E}_\mu s$ is a term of type σ.

Hence, $\mathrm{C}_{\mu\alpha\rho}$ is not a constant but a term former (a unary function symbol) and the infix notation $r\mathrm{E}_\mu s$ is used in place of $\mathrm{Rec}_{\mu\alpha\rho}sr$.

The *lifting* $\rho[\cdot]$ of $\lambda\alpha\rho$ to terms deserves some attention:

Definition 2.4. Instead of defining a term $\rho[r]$ of type $\rho[\alpha := \sigma] \to \rho[\alpha := \tau]$ for $r : \sigma \to \tau$ we define a closed term $\mathsf{lift}^+_{\lambda\alpha\rho}$ of type $\forall\alpha\forall\beta.(\alpha \to \beta) \to \rho \to \rho[\alpha := \beta]$ (for $\beta \notin \{\alpha\} \cup \mathsf{FV}(\rho)$) whenever $\mu\alpha\rho$ is in NPTy, i. e., $\rho \in \mathsf{NPTy}$ and $\alpha \in \mathsf{NPos}(\rho)$. Because positivity and negativity are defined simultaneously, we have to define in parallel auxiliary terms $\mathsf{lift}^-_{\lambda\alpha\rho} : \forall\alpha\forall\beta.(\alpha \to \beta) \to \rho[\alpha := \beta] \to \rho$ for $\rho \in \mathsf{NPTy}$ and $\alpha \in \mathsf{NNeg}(\rho)$. The definition is straightforward by structural induction on ρ thanks to the absence of interleaving. Throughout we assume that the variable f has type $\alpha \to \beta$ (corresponding to the functional to be lifted).

(triv) If $\alpha \notin \mathsf{FV}(\rho)$, then $\mathsf{lift}^+_{\lambda\alpha\rho} := \mathsf{lift}^-_{\lambda\alpha\rho} := \Lambda\alpha\Lambda\beta\lambda f\lambda x^\rho x$. All the other cases are under the proviso "otherwise" (this is why there is no clause pertaining to μ).

(V) $\mathsf{lift}^+_{\lambda\alpha\alpha} := \Lambda\alpha\Lambda\beta\lambda f f$. (Because $\alpha \notin \mathsf{NNeg}(\alpha)$, there is no clause for $\mathsf{lift}^-_{\lambda\alpha\alpha}$.)

($\times$) $\mathsf{lift}^+_{\lambda\alpha.\rho_1\times\rho_2} := \Lambda\alpha\Lambda\beta\lambda f\lambda x^{\rho_1\times\rho_2}.\Big\langle \mathsf{lift}^+_{\lambda\alpha\rho_1}\alpha\beta f(x\mathsf{L}), \mathsf{lift}^+_{\lambda\alpha\rho_2}\alpha\beta f(x\mathsf{R})\Big\rangle$.

$\mathsf{lift}^-_{\lambda\alpha.\rho_1\times\rho_2} := \Lambda\alpha\Lambda\beta\lambda f\lambda x^{(\rho_1\times\rho_2)[\alpha:=\beta]}.\Big\langle \mathsf{lift}^-_{\lambda\alpha\rho_1}\alpha\beta f(x\mathsf{L}), \mathsf{lift}^-_{\lambda\alpha\rho_2}\alpha\beta f(x\mathsf{R})\Big\rangle$.

($\to$) $\mathsf{lift}^+_{\lambda\alpha.\rho_1\to\rho_2} := \Lambda\alpha\lambda\beta\lambda f\lambda x^{\rho_1\to\rho_2}\lambda y^{\rho_1[\alpha:=\beta]}.\mathsf{lift}^+_{\lambda\alpha\rho_2}\alpha\beta f\Big(x(\mathsf{lift}^-_{\lambda\alpha\rho_1}\alpha\beta f y)\Big)$.

$\mathsf{lift}^-_{\lambda\alpha.\rho_1\to\rho_2} := \Lambda\alpha\lambda\beta\lambda f\lambda x^{(\rho_1\to\rho_2)[\alpha:=\beta]}\lambda y^{\rho_1}.\mathsf{lift}^-_{\lambda\alpha\rho_2}\alpha\beta f\Big(x(\mathsf{lift}^+_{\lambda\alpha\rho_1}\alpha\beta f y)\Big)$.

($\forall$) $\mathsf{lift}^+_{\lambda\alpha\forall\gamma\tau} := \Lambda\alpha\Lambda\beta\lambda f\lambda x^{\forall\gamma\tau}\Lambda\gamma.\mathsf{lift}^+_{\lambda\alpha\tau}\alpha\beta f(x\gamma)$.
$\mathsf{lift}^-_{\lambda\alpha\forall\gamma\tau} := \Lambda\alpha\Lambda\beta\lambda f\lambda x^{\forall\gamma\tau[\alpha:=\beta]}\Lambda\gamma.\mathsf{lift}^-_{\lambda\alpha\tau}\alpha\beta f(x\gamma)$.
(We may assume that $\gamma \neq \beta$.)

A definition which yields normal terms is shown in [7] pp. 73–75. Because we do not allow for interleaving one only has to omit the case (μ). The reader also finds a definition in [6] p. 311. Again one has to omit the μ-clause. Another place is [3] p. 206 which essentially contains the definitions. Note that with interleaving the definition would be much more involved as shown in [7] p. 78.

We are now in the position to extend $\mapsto$ by beta reduction for inductive types:

$$(\beta_\mu) \quad (\mathrm{C}_{\mu\alpha\rho}t)\mathrm{E}_\mu s \mapsto s\Big(\mathsf{lift}^+_{\lambda\alpha\rho}(\mu\alpha\rho)(\mu\alpha\rho\times\sigma)(\lambda x^{\mu\alpha\rho}.\langle x, (\lambda x^{\mu\alpha\rho}.x\mathrm{E}_\mu s)x\rangle)t\Big).$$

The reason for writing $(\lambda x^{\mu\alpha\rho}.x\mathrm{E}_\mu s)x$ instead of $x\mathrm{E}_\mu s$ is only technical as will be clear from the following

Example 2.5. Gödel's system T in the variant with the initial term and the step term as indices of the recursor may be *embedded* into NPI: Some calculation will show that

$$\mathsf{lift}^+_{\lambda\alpha.1+\alpha} \to^* \Lambda\alpha\Lambda\beta\lambda f^{\alpha\to\beta}\lambda x^{1+\alpha}\Lambda\delta\lambda z^{1\to\delta}\lambda z_1^{\beta\to\delta}.x\delta z(\lambda z_2^\alpha.z_1(fz_2)).$$

Define

$$\begin{array}{rcl} 0 & := & \mathrm{C}_{\mathsf{nat}}(\Lambda\delta\lambda x^{1\to\delta}\lambda y^{\mathsf{nat}\to\delta}.x(\Lambda\gamma\lambda z^{\gamma}z)) : \mathsf{nat} \\ \mathsf{S} & := & \lambda z^{\mathsf{nat}}.\mathrm{C}_{\mathsf{nat}}(\Lambda\delta\lambda x^{1\to\delta}\lambda y^{\mathsf{nat}\to\delta}.yz) : \mathsf{nat}\to\mathsf{nat} \end{array}$$

Given terms $a:\sigma$ and $b:\mathsf{nat}\to\sigma\to\sigma$ we define $R_{a,b} := \lambda x^{\mathsf{nat}}.x\mathrm{E}_\mu s : \mathsf{nat}\to\sigma$ with $s := \lambda x^{1+\mathsf{nat}\times\sigma}.x\sigma(\lambda y^1.a)(\lambda y^{\mathsf{nat}\times\sigma}.b(y\mathsf{L})(y\mathsf{R}))$, where $x,y\notin \mathsf{FV}(a)\cup\mathsf{FV}(b)$. We get the following reduction behaviour: $R_{a,b}0 \to^+ a$ and $R_{a,b}(\mathsf{S}t)\to^+ bt(R_{a,b}t)$, where the term $R_{a,b}$ is indeed a subterm of the reduct. Therefore, primitive recursion on naturals in all finite types is present in our system. Note that the embedding would be easier to read if there were sum types in our system. They are coded due to problems with Lemma 3.6 we want to avoid.

An elaborate comment concerning the relation to iteration is in order. Assuming a term s of type $\rho[\alpha:=\sigma]\to\sigma$ we could introduce a term $\mathrm{It}_{\mu\alpha\rho}s$ such that the diagram in figure 2 commutes.

$$\begin{tikzcd} \rho[\alpha:=\mu\alpha\rho] \arrow[r, "\mathrm{C}_{\mu\alpha\rho}"] \arrow[d, "{\rho[\mathrm{It}_{\mu\alpha\rho}s]}"'] & \mu\alpha\rho \arrow[d, "\mathrm{It}_{\mu\alpha\rho}s"] \\ \rho[\alpha:=\sigma] \arrow[r, "s"'] & \sigma \end{tikzcd}$$

FIGURE 2. iteration

More precisely, keep (μ-I) but replace the term formation rule (μ-E) by

$(\mu\text{-E})^i$ If r is a term of type $\mu\alpha\rho$ and s is a term of type $\rho[\alpha:=\sigma]\to\sigma$, then $r\mathrm{E}^i_\mu s$ is a term of type σ.

The beta rule becomes

$$(\beta_\mu)^i \quad (\mathrm{C}_{\mu\alpha\rho}t)\mathrm{E}^i_\mu s \mapsto s\Big(\mathsf{lift}^+_{\lambda\alpha\rho}(\mu\alpha\rho)\sigma(\lambda x^{\mu\alpha\rho}.x\mathrm{E}^i_\mu s)t\Big).$$

However, it is well-known that this reduction may be simulated within system F: If ρ is mapped to ρ', then $\mu\alpha\rho$ will be mapped to $(\mu\alpha\rho)' := \forall\alpha.(\rho'\to\alpha)\to\alpha$. If σ, r and s are already mapped to σ', r' and s', respectively, then $r\mathrm{E}^i_\mu s$ is mapped to $r'\sigma's'$. (In a sense, this easy encoding is built into $(\mu\alpha\rho)'$.) Finally, if $t:\rho[\alpha:=\mu\alpha\rho]$ is already mapped to $t':\rho'[\alpha:=(\mu\alpha\rho)']$, then $\mathrm{C}_{\mu\alpha\rho}t$ is mapped to

$$\Lambda\alpha\lambda z^{\rho'\to\alpha}.z\Big(\mathsf{lift}^+_{\lambda\alpha\rho'}(\mu\alpha\rho)'\alpha(\lambda x^{(\mu\alpha\rho)'}.x\alpha z)t'\Big).$$

The translation of nat would be $\mathsf{nat}' := \forall\alpha.((1+\alpha)\to\alpha)\to\alpha$. Intuitively, the type $((1+\alpha)\to\alpha)\to\alpha$ is isomorphic to $\alpha\to(\alpha\to\alpha)\to\alpha$. In order to get rid of the coded sum type, we only show how the type $\mathsf{nat}_{\mathsf{F}} := \forall\alpha.\alpha\to(\alpha\to\alpha)\to\alpha$

models iteration on naturals (we closely follow [5] section 11.5.1). Set

$$\begin{array}{lcl} 0_{\mathsf{F}} & := & \Lambda\alpha\lambda y^{\alpha}\lambda z^{\alpha\to\alpha}.y : \mathsf{nat}_{\mathsf{F}} \\ \mathsf{S}_{\mathsf{F}} & := & \lambda x^{\mathsf{nat}_{\mathsf{F}}}\Lambda\alpha\lambda y^{\alpha}\lambda z^{\alpha\to\alpha}.z(x\alpha yz) : \mathsf{nat}_{\mathsf{F}} \to \mathsf{nat}_{\mathsf{F}}. \end{array}$$

If $a : \sigma$ and $b : \sigma \to \sigma$, then $0_{\mathsf{F}}\sigma ab \to^{+} a$ and $(\mathsf{S}_{\mathsf{F}}t)\sigma ab \to^{+} b(t\sigma ab)$. Hence, $r\sigma ab$ describes the function defined by iteration with initial value a and step function b at the argument r. Modulo the sum types this is also the behaviour of $(\beta_{\mu})^{i}$ for nat. (Clearly, the pair of a and b corresponds to $s^{(1+\sigma)\to\sigma}$.)

Why do we get iteration via the above encoding? Set $\underline{n} := \underbrace{\mathsf{S}_{\mathsf{F}}(\dots(\mathsf{S}_{\mathsf{F}}}_{n}\, 0_{\mathsf{F}})\dots)$ for $n \in \mathbb{N}$. Clearly, $\underline{n} \to^{*} \Lambda\alpha\lambda y^{\alpha}\lambda z^{\alpha\to\alpha}.\underbrace{z(\dots(z}_{n}\, y)\dots)$. Hence, the underlying untyped lambda term of the numeral $\underline{n}$ is simply the n-th Church numeral and by itself the iterator.

The term $R_{a,b}$ from the previous example was more powerful in that b had the type $\mathsf{nat} \to \sigma \to \sigma$ which allowed to access also the argument in the recursion. We may try to produce an analogous term $R'_{a,b}$ within system F. The idea is to define the identity on $\mathsf{nat}_{\mathsf{F}}$ and the sought functional simultaneously by iteration: $R'_{a,b} := \lambda x^{\mathsf{nat}_{\mathsf{F}}}.x\sigma a'b'\mathsf{R}$ with $a' := \langle 0_{\mathsf{F}}, a\rangle$ and $b' := \lambda z^{\mathsf{nat}_{\mathsf{F}}\times\sigma}.\langle \mathsf{S}_{\mathsf{F}}(z\mathsf{L}), b(z\mathsf{L})(z\mathsf{R})\rangle$. By induction on n we may show that $\underline{n}\sigma a'b'\mathsf{L} =_{\beta} \underline{n}$ and $R'_{a,b}\underline{n} =_{\beta} R_{a,b}\underline{n}$. Neither do we see the correct reduction behaviour (only the correct equality), nor do we get the desired equality for arbitrary terms of the form $\mathsf{S}_{\mathsf{F}}t$. One might be interested in numerals only. However, already the type $\mathsf{tree}(\mathsf{nat})$ cannot be understood by confining the study to canonical inhabitants. Moreover—as pointed out in [5]—the iteratively defined predecessor function would be linear in the input number!

It is generally believed that there will not be any other computable encoding of nat in system F such that primitive recursion is simulated by F's rewrite rules.

2.3. Non-interleaving positive coinductive types

For the system NPC we simply have to dualize the defining diagram of NPI. This time we use the binder ν to form coinductive types of the form $\nu\alpha\rho$ if α occurs only positively in ρ and not free in any subexpression of $\nu\alpha\rho$ of the form $\nu\alpha'\rho'$. The type system hence is the same as for NPI but with a different name for the binder. The intuition behind $\nu\alpha\rho$, however, is the greatest fixed-point of the mapping $\sigma \mapsto \rho[\alpha := \sigma]$.

Full primitive corecursion is again introduced as in [3] (now taking the motivation from final coalgebras of functors): Assume a (legal) type $\nu\alpha\rho$ and a term s of type $\sigma \to \rho[\alpha := \nu\alpha\rho + \sigma]$ (we encode sum types impredicatively as in NPI). Then as a first approximation to the syntax of NPC we postulate the existence of a term $\mathrm{CoRec}_{\nu\alpha\rho}s$ making the diagram in figure 3 commute. ($\mathrm{E}_{\nu\alpha\rho}$ "unfolds" $\nu\alpha\rho$ into $\rho[\alpha := \nu\alpha\rho]$ and establishes part of the intuitive isomorphism $\rho[\alpha := \nu\alpha\rho] \cong \nu\alpha\rho$. Id, composition and $\rho[\cdot]$ are as before. $[\cdot,\cdot]$ denotes case distinction coded impredicatively.)

$$
\begin{tikzcd}
\rho[\alpha := \nu\alpha\rho] & \nu\alpha\rho \arrow[l, "\mathrm{E}_{\nu\alpha\rho}"'] \\
\rho[\alpha := \nu\alpha\rho + \sigma] \arrow[u, "{\rho[[\mathrm{Id}, \mathrm{CoRec}_{\nu\alpha\rho} s]]}"] & \sigma \arrow[l, "s"] \arrow[u, "\mathrm{CoRec}_{\nu\alpha\rho} s"']
\end{tikzcd}
$$

FIGURE 3. (full) primitive corecursion

We would have a constant $\mathrm{E}_{\nu\alpha\rho}$ of type $\nu\alpha\rho \to \rho[\alpha := \nu\alpha\rho]$, a term former $\mathrm{CoRec}_{\nu\alpha\rho}$ such that for terms s of type $\sigma \to \rho[\alpha := \nu\alpha\rho + \sigma]$, $\mathrm{CoRec}_{\nu\alpha\rho} s$ is a term of type $\sigma \to \nu\alpha\rho$, and the equality axiom

$$\mathrm{E}_{\nu\alpha\rho} \circ (\mathrm{CoRec}_{\nu\alpha\rho} s) = \rho[[\mathrm{Id}, \mathrm{CoRec}_{\nu\alpha\rho} s]] \circ s.$$

Again, we prefer another syntax for NPC and extend the definition of the *term system* of F by:

(ν-E) If r is a term of type $\nu\alpha\rho$, then $r\mathrm{E}_\nu$ is a term of type $\rho[\alpha := \nu\alpha\rho]$.

(ν-I) If s is a term of type $\sigma \to \rho[\alpha := \nu\alpha\rho + \sigma]$ and t is a term of type σ, then $\Omega_{\nu\alpha\rho} st$ is a term of type $\nu\alpha\rho$.

Hence, we adopt a postfix notation $r\mathrm{E}_\nu$ instead of the application of a constant $\mathrm{E}_{\nu\alpha\rho}$ to r and introduce $\Omega_{\nu\alpha\rho}$ as binary function symbol with result type $\nu\alpha\rho$ instead of the unary $\mathrm{CoRec}_{\nu\alpha\rho}$ of function type. (ν-E) is an elimination rule for $\nu\alpha\rho$ because a term of this type is fed in and (ν-I) is an introduction rule for $\nu\alpha\rho$ because a term of this type is generated.

The terms $\mathsf{lift}^+_{\lambda\alpha\rho}$ and $\mathsf{lift}^-_{\lambda\alpha\rho}$ are defined as before. (They are the same modulo the new name ν for the binder because of the absence of interleaving which allows to define the terms without reference to the new term rules.)

The relation $\mapsto$ of F is extended by beta conversion (β_ν) for coinductive types as follows:

$$(\Omega_{\nu\alpha\rho} st)\mathrm{E}_\nu \mapsto \mathsf{lift}^+_{\lambda\alpha\rho}(\nu\alpha\rho + \sigma)(\nu\alpha\rho)\Big(\lambda z^{\nu\alpha\rho+\sigma}.z(\nu\alpha\rho)\mathrm{Id}(\lambda x^\sigma.\Omega_{\nu\alpha\rho} sx)\Big)(st).$$

Note that the impredicative encoding of $\nu\alpha\rho + \sigma$ enters the definition when the variable z of sum type gets a type and two terms as arguments.

Example 2.6. A typical example would be the type $\mathsf{stream} := \nu\alpha.\rho \times \alpha$ (with $\alpha \notin \mathsf{FV}(\rho)$ for some fixed type ρ) modeling streams of elements of ρ. Why streams? Because we can associate with any term r of type stream an infinite sequence $(r_n)_{n\in\mathbb{N}}$ of terms of type ρ. Simply set $r_n := r\,\underbrace{\mathrm{E}_\nu\mathsf{R}\ldots\mathrm{E}_\nu\mathsf{R}}_{n}\,\mathrm{E}_\nu\mathsf{L}$. We first study coiteration. For this we keep (ν-E) and change (ν-I) to

(ν-I)^i If s is a term of type $\sigma \to \rho[\alpha := \sigma]$ and t is a term of type σ, then $\Omega^i_{\nu\alpha\rho} st$ is a term of type $\nu\alpha\rho$.

The beta rule becomes

$$(\beta_\nu)^i \quad (\Omega^i_{\nu\alpha\rho} st)\mathrm{E}_\nu \mapsto \mathsf{lift}^+_{\lambda\alpha\rho}\sigma(\nu\alpha\rho)(\lambda x^\sigma.\Omega_{\nu\alpha\rho} sx)(st).$$

We have $\mathsf{lift}^+_{\lambda\alpha.\rho\times\alpha} = \Lambda\alpha\Lambda\beta\lambda f^{\alpha\to\beta}\lambda x^{\rho\times\alpha}.\langle x\mathsf{L}, f(x\mathsf{R})\rangle$. Therefore, our coiteration rule $(\beta_\nu)^i$ gives us

$$(\Omega^i_{\mathsf{stream}} s^{\sigma\to\rho\times\sigma} t^\rho)\mathrm{E}_\nu \to^+ \langle st\mathsf{L}, \Omega^i_{\mathsf{stream}} s(st\mathsf{R})\rangle.$$

We now exploit the type isomorphism between $\sigma \to \rho\times\sigma$ and $(\sigma\to\rho)\times(\sigma\to\sigma)$ and change the syntax to

(s-I) If s_1 is a term of type $\sigma \to \rho$, s_2 is a term of type $\sigma\to\sigma$ and t is a term of type σ, then $\Omega s_1 s_2 t$ is a term of type stream.

(β_{s}) $(\Omega s_1 s_2 t)\mathrm{E}_\nu \mapsto \langle s_1 t, \Omega s_1 s_2(s_2 t)\rangle$

This will give in essence the same behaviour of stream. Now, if $r := \Omega s_1 s_2 t$, then $r_n \to^* s_1(\underbrace{s_2(\dots(s_2}_{n} t)\dots))$.

Interpretation: The type σ represents the internal state space, s_1 is the output function and s_2 the transition function while t is the initial state.

In [3] we find a general translation of coiteration (for "positive type schemes") into system F: If ρ is already translated into ρ', then $\nu\alpha\rho$ is translated into

$$(\nu\alpha\rho)' := \forall\beta.(\forall\alpha.(\alpha\to\rho')\to\alpha\to\beta)\to\beta.$$

For our example with (β_{s}) we may modify this to

$$\mathsf{stream}' := \forall\beta.\Big(\forall\alpha.(\alpha\to\rho')\to(\alpha\to\alpha)\to\alpha\to\beta\Big)\to\beta,$$

$$(\Omega s_1 s_2 t)' := \Lambda\beta\lambda z^{\forall\alpha.(\alpha\to\rho')\to(\alpha\to\alpha)\to\alpha\to\beta}.z\sigma' s_1' s_2' t' \qquad \text{and}$$

$$(r^{\mathsf{stream}}\mathrm{E}_\nu)' := r'(\rho'\times\mathsf{stream}')\Big(\Lambda\alpha\lambda y_1^{\alpha\to\rho'}\lambda y_2^{\alpha\to\alpha}\lambda z^\alpha.\langle y_1 z, \Big(\Omega y_1 y_2(y_2 z)\Big)'\rangle\Big).$$

Clearly, the primed types and terms are assumed to be already translated. Note that the last clause refers to the last but one. All the other type rules and term rules shall be given homomorphically.

It is easy to see that $(\Omega s_1 s_2 t\mathrm{E}_\nu)' \to^+ \langle s_1 t, \Omega s_1 s_2(s_2 t)\rangle'$. This shows the embedding of coiteration into F for this example.

An example for the use of coiteration would be with $\sigma := \mathsf{stream}\times\mathsf{stream}$, $s_1 := \lambda x^\sigma.x\mathsf{L}\mathrm{E}_\nu\mathsf{L} : \sigma\to\rho$ and $s_2 := \lambda x^\sigma.\langle x\mathsf{R}, x\mathsf{L}\mathrm{E}_\nu\mathsf{R}\rangle : \sigma\to\sigma$. Set $r := \Omega s_1 s_2\langle a, b\rangle$ for $a, b : \mathsf{stream}$. Then $r_{2n} \to^* a_n$ and $r_{2n+1} \to^* b_n$.

For corecursion we similarly change the syntax to

$(\text{s-I})^r$ If s_1 is a term of type $\sigma\to\rho$, s_2 is a term of type $\sigma\to\mathsf{stream}+\sigma$ and t is a term of type σ, then $\Omega^r s_1 s_2 t$ is a term of type stream.

$(\beta_{\mathsf{s}})^r$ $(\Omega^r s_1 s_2 t)\mathrm{E}_\nu \mapsto \langle s_1 t, s_2 t\ \mathsf{stream}(\lambda x^{\mathsf{stream}}x)(\lambda x^\sigma.\Omega^r s_1 s_2 x)\rangle$.

Hence, if $s_2 = \lambda y^\sigma \Lambda\beta\lambda z_1^{\mathsf{stream}\to\beta}\lambda z_2^{\sigma\to\beta}.z_2(\hat{s}_2 y)$ for some $\hat{s}_2 : \sigma \to \sigma$, i. e., for s_2 being the composition of the encoded right injection with $\hat{s}_2$, we get the reduction behaviour of iteration.

However, we may not only calculate a new state via s_2 but also initiate an arbitrary stream (in case s_2 applied to the actual state happens to be in the stream part of $\mathsf{stream} + \sigma$): "The machine executes a new program." In [3] we find a very simple example with corecursion for inhabited types ρ, i. e., if there is a closed term $a : \rho$. Set $\sigma := \mathsf{stream}$, let $t : \sigma$. Set

$$r := \lambda x^\sigma.\Omega^r(\lambda y^\sigma.a)\Big(\lambda y^\sigma \Lambda\beta\lambda z_1^{\mathsf{stream}\to\beta}\lambda z_2^{\sigma\to\beta}.z_1(y\mathrm{E}_\nu \mathsf{R})\Big)x.$$

The second argument to Ω^r is the composition of the left injection into our sum type with $\lambda y^{\mathsf{stream}}.y\mathrm{E}_\nu\mathsf{R}$. Then $(rt)_0 \to^* a$ and for all n we have $(rt)_{n+1} \to^* t_{n+1}$. Hence, rt outputs the same stream as t but for the first element which is set to a.

If in general corecursion could be embedded into coiteration, then also into system F and by the results of this paper (and also already by [3]) recursion would also embed into system F and this is extremely unlikely.

2.4. Monotone (co)inductive types

We now abstract away from the canonically defined terms $\mathsf{lift}^+_{\lambda\alpha\rho}$ whose types $\forall\alpha\forall\beta.(\alpha\to\beta)\to\rho\to\rho[\alpha:=\beta]$ simply express the monotonicity of $\lambda\alpha\rho$ internally. Their existence previously has been guaranteed by the syntactic condition of positivity and absence of interleaving (the second of which is for reasons of simplicity). We drop these conditions on formation of $\mu\alpha\rho$ and $\nu\alpha\rho$ and instead require a monotonicity witness in the term formation rules (μ-E) and (ν-I) where a monotonicity witness is simply an already generated term in the system having type $\forall\alpha\forall\beta.(\alpha\to\beta)\to\rho\to\rho[\alpha:=\beta]$. This term is then used for the formulation of the beta reduction rules (β_μ) and (β_ν).

In this manner we arrive at the systems MI and MC of monotone inductive types and of monotone coinductive types. More precisely, the types of MI are defined by adding the quantifier μ (without restriction—i. e., any $\mu\alpha\rho$ is a type if ρ is a type and α is a type variable) to the type system of F. The term formation rules of F are supplemented by

(μ-I) As for system NPI.

(μ-E) If $r : \mu\alpha\rho$, $m : \forall\alpha\forall\beta.(\alpha\to\beta)\to\rho\to\rho[\alpha:=\beta]$ and s is a term of type $\rho[\alpha:=\mu\alpha\rho\times\sigma]\to\sigma$, then $r\mathrm{E}_\mu ms$ is a term of type σ.

The new beta reduction clause:

$$(\beta_\mu)\quad (\mathrm{C}_{\mu\alpha\rho}t)\mathrm{E}_\mu ms \mapsto s\Big(m(\mu\alpha\rho)(\mu\alpha\rho\times\sigma)(\lambda x^{\mu\alpha\rho}.\langle x,(\lambda x^{\mu\alpha\rho}.x\mathrm{E}_\mu ms)x\rangle)t\Big).$$

The witnessing term m need not be closed. If m is open, this becomes conditional monotonicity or even hypothetical monotonicity (in case m is a variable). More interesting are non-positive $\lambda\alpha\rho$ with closed witnesses as in the following

Example 2.7. As an example for a monotone inductive type we give

$$\mu\alpha.1 + ((((\alpha \to \rho) \to \alpha) \to \alpha) \to \alpha)$$

for $\alpha \notin \mathsf{FV}(\rho)$ (an invention by U. Berger). It is quite easy to find a monotonicity witness for $\lambda\alpha.1 + ((((\alpha \to \rho) \to \alpha) \to \alpha) \to \alpha$ although we do not have a positive dependency due to the last but one occurrence of α (we omit type information):

$$\Lambda\alpha\Lambda\beta\lambda f\lambda x\Lambda\gamma\lambda z\lambda z_1.x\gamma z(\lambda z_2.z_1(\lambda z_3.z_3(\lambda z_4.f(z_2(\lambda z_5.z_5(\lambda z_6.z_4(fz_6))))))).$$

The types of MC are the same as those of MI but with the binder μ replaced by ν. The corresponding term rules are as follows:

(ν-E) As for system NPC.
(ν-I) If $m : \forall\alpha\forall\beta.(\alpha \to \beta) \to \rho \to \rho[\alpha := \beta]$, $s : \sigma \to \rho[\alpha := \nu\alpha\rho + \sigma]$ and t is a term of type σ, then $\Omega_{\nu\alpha\rho}mst$ is a term of type $\nu\alpha\rho$.

Accordingly, the rule (β_ν) of corecursion is

$$(\Omega_{\nu\alpha\rho}mst)\mathrm{E}_\nu \mapsto m(\nu\alpha\rho + \sigma)(\nu\alpha\rho)\Big(\lambda z^{\nu\alpha\rho+\sigma}.z(\nu\alpha\rho)\mathrm{Id}(\lambda x^\sigma.\Omega_{\nu\alpha\rho}msx)\Big)(st).$$

2.5. Non-interleaving positive fixed-point types

The idea (according to [3], p. 214, due to C. Paulin-Mohring) is to take the simple term formation rules (μ-I) and (ν-E), merge $\mu\alpha\rho$ and $\nu\alpha\rho$ to $f\alpha\rho$, and leave out minimality and maximality altogether. The system NPF has the same type system as NPI but with binder f (for fixed-point) instead of μ, i.e., $f\alpha\rho$ is only allowed if α occurs only positively in ρ and there is no interleaving. The term rules of NPF extend F by:

(f-I) $(\mathrm{C}_{f\alpha\rho}t^{\rho[\alpha:=f\alpha\rho]}) : f\alpha\rho$
(f-E) $(r^{f\alpha\rho}\mathrm{E}_f) : \rho[\alpha := f\alpha\rho]$

The beta rule expresses one half of the intuition that $f\alpha\rho$ denotes a fixed-point of $\sigma \mapsto \rho[\alpha := \sigma]$:

$$(\beta_f) \quad (\mathrm{C}_{f\alpha\rho}t)\mathrm{E}_f \mapsto t.$$

As we only study this additional rule the name retract types as used in [3] would be more appropriate. However, the canonical eta rule would be

$$(\eta_f) \quad \mathrm{C}_{f\alpha\rho}(r\mathrm{E}_f) \mapsto r,$$

reflecting the other half of the above intuition. (In [8] it is shown that the system with (η_f) is strongly normalizing.)

3. The embeddings

We have defined 5 extensions of system F (with eta rules) by type constructs, corresponding term rules and beta reduction rules designed for the study of intensional properties of fixed-points of monotone operators. We list them with a one line description.

NPF: fixed-point types $f\alpha\rho$ with α only positively in ρ and no interleaved f
NPI: inductive types $\mu\alpha\rho$ with α only positively in ρ and no interleaved μ
MI: inductive types with arbitrary monotonicity witnesses and interleaving
NPC: coinductive types $\nu\alpha\rho$ with α only positively in ρ and no interleaved ν
MC: coinductive types with arbitrary monotonicity witnesses and interleaving

(N = no interleaving, P = positivity, M = monotonicity witness, F = fixed-point, I = inductive, C = coinductive)

Theorem 3.1. *The systems NPF, NPI, MI, NPC, MC in the preceding section embed into each other.*

(See the definition of embedding in the introduction.)

The main corollary is strong normalization for all of them since NPF (and MI) has been shown to be strongly normalizing before (again refer to the introduction). Confluence for all of them holds but is no consequence of the embeddings (cf. the introduction).

The proof will be organized as follows:

- NPF embeds into NPC because the canonical corecursive definition of the constructor is well-behaved. In order to check that, we have to prove some restricted version of functoriality of the terms $\mathsf{lift}^{+}_{\lambda\alpha\rho}$ which in turn shows the necessity of eta rules and explains why we restrict to *non-interleaving* fixed-point types and do not include sum types explicitly in our systems.
- NPF embeds into NPI because we have a predecessor function (i. e., a destructor) in NPI and also some functoriality of $\mathsf{lift}^{+}_{\lambda\alpha\rho}$.
- NPC embeds into MC, and NPI embeds into MI, because we may simply take the canonical monotonicity witnesses $\mathsf{lift}^{+}_{\lambda\alpha\rho}$ as the monotonicity witness required in the term formation rules (μ-E) and (ν-I). We again profit from the absence of interleaving.
- MC and MI embed into NPF: We first embed the systems into the extension NPFex of NPF by the second-order existential quantifier. E. g., we translate the types of MC via:

  $$(\nu\alpha\rho)' := f\beta\exists\gamma.(\forall\alpha.(\beta+\gamma\to\alpha)\to\gamma\to\rho')\times\gamma,$$

  where ρ' is the translation of ρ. The additional quantifier for α does the positivization and removes the interleaving of type variables.

  The embedding of NPFex into NPF is done by the standard encoding of the second-order existential quantifier in system F.

3.1. Embedding NPF into NPC

In order to embed the system with fixed-point types into that of coinductive types, we have to encode the constructor via corecursion, i. e., we have to define a closed term $C : \rho[\alpha := \nu\alpha\rho] \to \nu\alpha\rho$ such that for every term t of type $\rho[\alpha := \nu\alpha\rho]$, we have that $Ct\mathrm{E}_\nu \to^+ t$. Writing

$$r := \mathsf{lift}^+_{\lambda\alpha\rho}(\nu\alpha\rho)(\nu\alpha\rho + \rho[\alpha := \nu\alpha\rho])(\lambda x^{\nu\alpha\rho}\Lambda\alpha\lambda y^{\nu\alpha\rho\to\alpha}\lambda z^{\rho[\alpha:=\nu\alpha\rho]\to\alpha}.yx)$$

which is nothing but the lifted left injection into the impredicatively encoded sum, the solution is $C := \lambda x^{\rho[\alpha:=\nu\alpha\rho]}.\Omega_{\nu\alpha\rho}rx$: We calculate (setting $\sigma := \rho[\alpha := \nu\alpha\rho]$)

$$\begin{aligned} Ct\mathrm{E}_\nu \quad \to_{\beta_\to}\to_{\beta_\nu} \quad & \mathsf{lift}^+_{\lambda\alpha\rho}(\nu\alpha\rho+\sigma)(\nu\alpha\rho)\Big(\lambda z^{\nu\alpha\rho+\sigma}.z(\nu\alpha\rho)\mathrm{Id}(\lambda x^\sigma.\Omega_{\nu\alpha\rho}rx)\Big) \\ & \Big(\mathsf{lift}^+_{\lambda\alpha\rho}(\nu\alpha\rho)(\nu\alpha\rho+\sigma)(\lambda x^{\nu\alpha\rho}\Lambda\alpha\lambda y^{\nu\alpha\rho\to\alpha}\lambda z^{\sigma\to\alpha}.yx)t\Big). \end{aligned}$$

We observe that

$$\Big(\lambda z^{\nu\alpha\rho+\sigma}.z(\nu\alpha\rho)\mathrm{Id}(\lambda x^\sigma.\Omega_{\nu\alpha\rho}rx)\Big) \circ \Big(\lambda x^{\nu\alpha\rho}\Lambda\alpha\lambda y^{\nu\alpha\rho\to\alpha}\lambda z^{\sigma\to\alpha}.yx\Big) = \mathrm{Id}_{\nu\alpha\rho},$$

where we write $=$ for the transitive, reflexive and symmetric closure of $\to$ which is the equality relation induced by $\to$. The equation does not come as a surprise since the left side is the composition of the left injection with a case construct which in the left case reproduces its argument. Clearly, we now want $\mathsf{lift}^+_{\lambda\alpha\rho}$ to fulfill the functor laws, i. e., it should "map the identity to the identity and commute with composition". For our purposes we do not need full functoriality but we have to consider reduction instead of the induced equality relation.

Lemma 3.2 (Restricted functoriality for sums)**.**

$$\mathsf{lift}^+_{\lambda\alpha\rho}(\sigma+\tau)\sigma(\lambda z^{\sigma+\tau}.z\sigma\mathrm{Id}_\sigma r)\Big(\mathsf{lift}^+_{\lambda\alpha\rho}\sigma(\sigma+\tau)(\lambda x^\sigma\Lambda\alpha\lambda y^{\sigma\to\alpha}\lambda z^{\tau\to\alpha}.yx)t\Big) \to^* t$$

for any types σ and τ and terms $r : \tau \to \sigma$ and $t : \rho[\alpha := \sigma]$.

Proof. By an easy induction on ρ. Simultaneously one has to prove a similar statement on $\mathsf{lift}^-_{\lambda\alpha\rho}$ for α only negative in ρ, namely

$$\mathsf{lift}^-_{\lambda\alpha\rho}\sigma(\sigma+\tau)(\lambda x^\sigma\Lambda\alpha\lambda y^{\sigma\to\alpha}\lambda z^{\tau\to\alpha}.yx)\Big(\mathsf{lift}^-_{\lambda\alpha\rho}(\sigma+\tau)\sigma(\lambda z^{\sigma+\tau}.z\sigma\mathrm{Id}_\sigma r)t\Big) \to^* t.$$

□

It is crucial that we included eta rules in our system. Moreover, if we had sum types directly in our system, we would have to include permutative conversions for them in order to cover the case that ρ is a sum type. Although permutative conversions do no harm to strong normalization, the proof of this fact has some subtleties in it (see [9] for the case of simply-typed lambda-calculus). If we allowed interleaving, it would be even worse: We would be in need of permutative conversions for coinductive types which seem to be completely unknown.

As an instance of the lemma we get that C serves as a constructor.

Definition 3.3 (Embedding NPF into NPC)**.** Let ρ' be ρ after replacing every binder f by ν. Obviously, $(\rho[\alpha := \sigma])' = \rho'[\alpha := \sigma']$ and $\mathsf{FV}(\rho') = \mathsf{FV}(\rho)$. For every term r^ρ of system NPF define the term r' of system NPC by recursion on r and simultaneously prove (the proofs are omitted due to their simplicity) that $r' : \rho'$ and $\mathsf{FTV}(r') = \mathsf{FTV}(r)$ and $\mathsf{FV}(r') = \{x^{\sigma'} | x^\sigma \in \mathsf{FV}(r)\}$ as follows:

(V) $(x^\rho)' := x^{\rho'}$.
(×-I) $\langle r, s\rangle' := \langle r', s'\rangle$.
(×-E) $(r\mathsf{L})' := r'\mathsf{L}$. $(r\mathsf{R})' := r'\mathsf{R}$.
(→-I) $(\lambda x^\rho r)' := \lambda x^{\rho'} r'$, where we may assume due to the renaming convention that for every $x^\sigma \in \mathsf{FV}(r)$ we have $\sigma = \rho$.
(→-E) $(rs)' := r's'$.
(∀-I) $(\Lambda\alpha r)' := \Lambda\alpha r'$. (Well-definedness follows from the claim on $\mathsf{FV}(r)$).
(∀-E) $(r\sigma)' := r'\sigma'$.
(f-I) $(\mathrm{C}_{f\alpha\rho} t)' := \Omega_{\nu\alpha\rho'} r t'$ with

$$r := \mathsf{lift}^+_{\lambda\alpha\rho'}(\nu\alpha\rho')(\nu\alpha\rho' + \rho'[\alpha := \nu\alpha\rho'])(\lambda x^{\nu\alpha\rho'}\Lambda\alpha\lambda y^{\nu\alpha\rho'\to\alpha}\lambda z^{\rho'[\alpha:=\nu\alpha\rho']\to\alpha}.yx).$$

(Recall that $\rho'[\alpha := \nu\alpha\rho'] = (\rho[\alpha := \nu\alpha\rho])')$.
(f-E) $(r^{f\alpha\rho}\mathrm{E}_f)' := r'\mathrm{E}_\nu$.

Lemma 3.4. *$(r[x^\rho := s])' = r'[x^{\rho'} := s']$ and $(r[\alpha := \sigma])' = r'[\alpha := \sigma']$.*

Proof. By induction on r. □

Lemma 3.5. *If $r \to \hat{r}$ in NPF, then $r' \to^+ \hat{r}'$ in NPC.*

Proof. First show it for $r \mapsto \hat{r}$, then infer it generally by help of the previous lemma. The rules $\beta_\to$ and $\beta_\forall$ are clear because of the previous lemma, the rules for product types are clear, the eta rules go through because of the properties of free type and term variables of r' proved with the definition of r'. Finally, β_f is dealt with in our introductory discussion of the constructor. □

3.2. Embedding NPF into NPI

The embedding of NPF into NPI is dual to the preceding embedding of NPF into NPC: For an embedding of the system with fixed-point types into that with inductive types, we have to define a (generalized) predecessor function (more precisely: a destructor) in NPI, i. e., a closed term P of type $\mu\alpha\rho \to \rho[\alpha := \mu\alpha\rho]$ such that $P(\mathrm{C}_{\mu\alpha\rho} t) \to^+ t$ for every term t of type $\rho[\alpha := \mu\alpha\rho]$. As a solution we set

$$P := \lambda x^{\mu\alpha\rho}.x\mathrm{E}_\mu\Big(\mathsf{lift}^+_{\lambda\alpha\rho}(\mu\alpha\rho \times \rho[\alpha := \mu\alpha\rho])(\mu\alpha\rho)(\lambda z^{\mu\alpha\rho\times\rho[\alpha:=\mu\alpha\rho]}.z\mathsf{L})\Big).$$

Again we need some restricted form of functoriality of $\mathsf{lift}^+_{\lambda\alpha\rho}$:

Lemma 3.6 (Restricted functoriality for products)**.**

$$\mathsf{lift}^+_{\lambda\alpha\rho}(\sigma\times\tau)\sigma(\lambda z^{\sigma\times\tau}.z\mathsf{L})\Big(\mathsf{lift}^+_{\lambda\alpha\rho}\sigma(\sigma\times\tau)(\lambda x^\sigma.\langle x,r\rangle)t\Big)\to^* t \quad \textit{and}$$

$$\mathsf{lift}^-_{\lambda\alpha\rho}\sigma(\sigma\times\tau)(\lambda x^\sigma.\langle x,r\rangle)\Big(\mathsf{lift}^-_{\lambda\alpha\rho}(\sigma\times\tau)\sigma(\lambda z^{\sigma\times\tau}.z\mathsf{L})t\Big)\to^* t$$

for $r:\tau$ and $t:\rho[\alpha:=\sigma]$ (and α positive in ρ for the statement on $\mathsf{lift}^+_{\lambda\alpha\rho}$ and α negative in ρ for that on $\mathsf{lift}^-_{\lambda\alpha\rho}$).

Proof. Again by induction on ρ, and an easy calculation shows that P is indeed a predecessor function. □

The formal embedding and its justification may be given in the same fashion as in the preceding section.

3.3. Embedding NPC into MC and NPI into MI

Clearly, one expects the embedding of positive (co)inductive types into monotone (co)inductive types to be an easy task. In our setting it is indeed very easy because there is no interleaving in NPC and NPI and hence the terms $\mathsf{lift}^+_{\lambda\alpha\rho}$ are built without the help of (ν-E), (ν-I), (μ-I) and (μ-E). Otherwise one would have to find an appropriate induction measure to define the embeddings (see [7] for the inductive case) but here recursion on the term structure suffices. We only consider the embedding of NPC into MC.

Definition 3.7 (Embedding of NPC into MC)**.** The embedding of the types will be the trivial one, i. e., $\rho' := \rho$ for every type ρ in NPC.

For every term r^ρ of system NPC define the term r' of MC by recursion on r and simultaneously prove (not shown) that $r':\rho$ and $\mathsf{FTV}(r')=\mathsf{FTV}(r)$ and $\mathsf{FV}(r')=\mathsf{FV}(r)$ as follows:

(F) The homomorphic term rules for F as in section 3.1. (Because of $\rho'=\rho$ the rule ($\to$-I) may be simplified to $(\lambda x^\rho r)' := \lambda x^\rho r'$ without the additional assumption.)
(ν-E) $(r\mathrm{E}_\nu)' := r'\mathrm{E}_\nu$.
(ν-I) $(\Omega_{\nu\alpha\rho}st)' := \Omega_{\nu\alpha\rho}\mathsf{lift}^+_{\lambda\alpha\rho}s't'$.

Note again that $\mathsf{lift}^+_{\lambda\alpha\rho}$ already is a term of MC because there is no interleaving. Therefore, $(\mathsf{lift}^+_{\lambda\alpha\rho})' = \mathsf{lift}^+_{\lambda\alpha\rho}$.

Lemma 3.8. *$(r[x^\rho:=s])' = r'[x^\rho:=s']$ and $(r[\alpha:=\sigma])' = r'[\alpha:=\sigma]$.*

Proof. By induction on r. In the crucial case (ν-I) we use that $\mathsf{lift}^+_{\lambda\alpha\rho}$ is closed and that $\mathsf{lift}^+_{\lambda\alpha\rho}[\beta:=\sigma] = \mathsf{lift}^+_{(\lambda\alpha\rho)[\beta:=\sigma]}$ which is easily proved by induction on ρ (and requires the analogous statement on $\mathsf{lift}^-_{\lambda\alpha\rho}$ to be proved simultaneously). □

It is now quite easy to verify that $'$ is indeed an embedding of NPC into MC. The embedding of NPI into MI is defined analogously.

3.4. Embedding MC and MI into NPF

This section is devoted to the proof of the collapse of monotone (co)inductive types into non-interleaving positive fixed-point types. It shows the strength of impredicative constructions.

In order to better display the construction for MC we define an embedding into the system NPFex which is NPF enriched with existential types. The standard encoding of existential types in system F with essential clause

$$(\exists\alpha\rho)' := \forall\beta.(\forall\alpha.\rho' \to \beta) \to \beta, \quad \beta \notin \{\alpha\} \cup \mathsf{FV}(\rho)$$

(see [5] p. 86 for the encoding and [7] for a careful proof in the style of section 3.1 that this indeed gives an embedding) trivially carries over to NPF (but not to NPI or NPC because formation of $\mathsf{lift}^+_{\lambda\alpha\rho}$ and the encoding of the existential do not commute!). Therefore, also NPFex embeds into NPF.

We define NPFex: The *type system* is that of NPF where instead of one unrestricted quantifier $\forall$ we have two of them: $\forall$ and $\exists$. The term formation rules now range over the extended type system. Moreover, we have the *new term formation rules*

($\exists$-I) If t is a term of type $\rho[\alpha := \tau]$, then $\mathrm{C}_{\exists\alpha\rho,\tau}t$ is a term of type $\exists\alpha\rho$.
($\exists$-E) If r is a term of type $\exists\alpha\rho$ and s is a term of type $\forall\alpha.\rho \to \sigma$ with $\alpha \notin \mathsf{FV}(\sigma)$, then $r\mathrm{E}_\exists s$ is a term of type σ.

This definition follows the standard natural deduction formulation of the second-order existential quantifier but with proof terms included.

The new beta rule is

$$(\beta_\exists) \quad \mathrm{C}_{\exists\alpha\rho,\tau}t\mathrm{E}_\exists s \mapsto s\tau t.$$

We now give the crucial clauses of the definition of ρ' for the two embeddings (all the other clauses are homomorphic):

(MC) $(\nu\alpha\rho)' := f\beta\exists\gamma.(\forall\alpha.(\beta + \gamma \to \alpha) \to \gamma \to \rho') \times \gamma$.
(MI) $(\mu\alpha\rho)' := f\beta\forall\gamma.(\forall\alpha.(\alpha \to \beta \times \gamma) \to \rho' \to \gamma) \to \gamma$.

(Of course, we choose $\beta \neq \gamma$ and $\beta, \gamma \notin \{\alpha\} \cup \mathsf{FV}(\rho)$.)

Clearly, these definitions yield types of system NPFex (and even in NPF for MI). In both cases we have a non-strict positive dependency on β (of the type's kernel).

Example 3.9. Setting $\sigma := 1 + ((((\alpha \to \rho) \to \alpha) \to \alpha) \to \alpha)$, we translate $\mu\alpha\sigma$: ρ' is the translation of ρ and has $\mathsf{FV}(\rho') = \mathsf{FV}(\rho)$ (see below). Our system F encodings of 1 and + are not affected by the embedding. Therefore,

$$(\mu\alpha\sigma)' = f\beta\forall\gamma.\Big(\forall\alpha.(\alpha \to \beta\times\gamma) \to \Big(1+((((\alpha \to \rho') \to \alpha) \to \alpha) \to \alpha)\Big) \to \gamma\Big) \to \gamma.$$

Because $\beta \neq \gamma$ and $\beta \notin \{\alpha\} \cup \mathsf{FV}(\rho')$, there is only one occurrence of β to look at. We reach β by passing $\forall\gamma$, going once to the left of $\to$, passing $\forall\alpha$, going again to the left of $\to$ and then to the right of $\to$ and finally to a part of $\times$. Two times left is non-strictly positive!

Note that we do not get dual constructions because we cannot set

$$(\mu\alpha\rho)' := f\beta\forall\gamma\exists\alpha.((\alpha \to \beta \times \gamma) \to \rho' \to \gamma) \to \gamma$$

which should be compared with the slight variant of (MC) which *would work*:

$$(\nu\alpha\rho)' := f\beta\exists\gamma\forall\alpha.((\beta + \gamma \to \alpha) \to \gamma \to \rho') \times \gamma.$$

This setting for $(\mu\alpha\rho)'$ would only classically give an isomorphic type to the previously defined (MI) but not constructively and—much more important—there is simply no reasonable clause for $(\mathrm{C}_{\mu\alpha\rho}t)'$ fitting to this definition.

Let us compare the definitions with those in [3] for embeddings of systems with positive type schemes (which are a restriction of non-interleaved positivity) instead of arbitrary monotonicity. In our notation they read:

(NPC) $(\nu\alpha\rho)' := f\beta\exists\gamma.(\gamma \to \rho'[\alpha := \beta + \gamma]) \times \gamma.$
(NPI) $(\mu\alpha\rho)' := f\beta\forall\gamma.(\rho'[\alpha := \beta \times \gamma] \to \gamma) \to \gamma.$

We recognize that the additional quantifier for α is responsible for the positivization and the removal of the interleaving of type variables.

For (MC) and (MI) we have that $(\rho[\alpha := \sigma])' = \rho'[\alpha := \sigma']$ and $\mathsf{FV}(\rho') = \mathsf{FV}(\rho)$.

Setting $\tau := (\forall\alpha.((\nu\alpha\rho)' + \gamma \to \alpha) \to \gamma \to \rho') \times \gamma$, the crucial clauses of the definition of r' for the embedding of MC into NPFex are :

$(\nu\text{-E})$ $(r\mathrm{E}_\nu)' := r'\mathrm{E}_f\mathrm{E}_\exists\Big(\Lambda\gamma\lambda u^\tau.u\mathsf{L}(\nu\alpha\rho)'$
$$\Big(\lambda z^{(\nu\alpha\rho)'+\gamma}.z(\nu\alpha\rho)'\mathrm{Id}_{(\nu\alpha\rho)'}\Big(\lambda x^\gamma.\mathrm{C}_{(\nu\alpha\rho)'}\mathrm{C}_{\exists\gamma\tau,\gamma}\langle u\mathsf{L}, x\rangle\Big)\Big)(u\mathsf{R})\Big).$$
$(\nu\text{-I})$ $(\Omega_{\nu\alpha\rho}mst)' := \mathrm{C}_{(\nu\alpha\rho)'}\mathrm{C}_{\exists\gamma\tau,\sigma'}\Big\langle\Lambda\alpha\lambda z^{(\nu\alpha\rho)'+\sigma'\to\alpha}\lambda x^{\sigma'}.m'(\nu\alpha\rho{+}\sigma)'\alpha z(s'x), t'\Big\rangle.$

The other rules are again the homomorphic ones, and we have to prove simultaneously with the definition that if $r : \rho$, then $r' : \rho'$, $\mathsf{FTV}(r') = \mathsf{FTV}(r)$ and $\mathsf{FV}(r') = \{x^{\sigma'} | x^\sigma \in \mathsf{FV}(r)\}$.

After having proved Lemma 3.4 for this situation the main task is to verify

$$(\Omega_{\nu\alpha\rho}mst\mathrm{E}_\nu)' \to^+ \Big(m(\nu\alpha\rho + \sigma)(\nu\alpha\rho)\Big(\lambda z^{\nu\alpha\rho+\sigma}.z(\nu\alpha\rho)\mathrm{Id}(\lambda x^\sigma.\Omega_{\nu\alpha\rho}msx)\Big)(st)\Big)'.$$

It is a calculation which is not affected by an unfortunate reduction strategy and therefore left to the reader.

For the embedding of MI into NPF the non-trivial clauses are

$(\mu\text{-I})$ $(\mathrm{C}_{\mu\alpha\rho}t)' := \mathrm{C}_{(\mu\alpha\rho)'}\Big(\Lambda\gamma\lambda u^{\forall\alpha.(\alpha\to(\mu\alpha\rho)'\times\gamma)\to\rho'\to\gamma}.u(\mu\alpha\rho)'$
$$\Big(\lambda x^{(\mu\alpha\rho)'}.\langle x, (\lambda x^{(\mu\alpha\rho)'}.x\mathrm{E}_f\gamma u)x\rangle\Big)t'\Big).$$
$(\mu\text{-E})$ $(r\mathrm{E}_\mu ms)' := r'\mathrm{E}_f\sigma'\Big(\Lambda\alpha\lambda v^{\alpha\to(\mu\alpha\rho)'\times\sigma'}\lambda w^{\rho'}.s'\Big(m'\alpha((\mu\alpha\rho)' \times \sigma')vw\Big)\Big).$

For showing that

$$(\mathrm{C}_{\mu\alpha\rho}t\mathrm{E}_\mu ms)' \to^+ \Big(s\Big(m(\mu\alpha\rho)(\mu\alpha\rho \times \sigma)(\lambda x^{\mu\alpha\rho}.\langle x, (\lambda x^{\mu\alpha\rho}.x\mathrm{E}_\mu ms)x\rangle)t\Big)\Big)'$$

we need six beta reduction steps. The usual machinery then allows to conclude that $'$ in fact embeds MI into NPF.

4. Concluding remarks

We have seen that the algorithmic weakness of system F w. r. t. (co)inductive data types (only iteration is modelled) is removed by adding fixed-points for non-interleaving positive $\lambda\alpha\rho$. Further extensions by stipulating weak initiality or weak finality even for interleaved use of parameters and arbitrary monotonicity witnesses could then be encoded in a reduction-preserving way and hence did not give additional algorithmic power.

Without any difficulty one could also combine NPI and NPC to a system of non-interleaving positive inductive and coinductive types. Due to the absence of interleaving this would hardly allow more than the study of hierarchical alternation of μ and ν (which is e. g. needed for the type $\nu\alpha.\mathsf{nat} \times \alpha$ of streams of naturals). But we may as well combine MI and MC where we e. g. may reason by coinduction when establishing the monotonicity of some inductive type. It is fairly obvious that only the embeddings shown for the constituent systems have to be merged in order to embed the combined systems e. g. into NPF.

The inclusion of product types into our base system F is only done for convenience. We could also take the standard impredicative encoding in the formulation of the beta rule of primitive recursion.

One may also study systems of interleaving positive inductive and coinductive types. A concise definition of the terms $\mathsf{lift}^{+}_{\lambda\alpha\rho}$ may only be given by help of iteration/coiteration which justifies to work with systems having both iteration and recursion (and coiteration and corecursion) as primitives (see [7] for the inductive case). Establishing functoriality properties of those $\mathsf{lift}^{+}_{\lambda\alpha\rho}$ is much harder and requires at least an eta rule for (co)iteration. For a proof of functoriality w. r. t. some parametric equality theory see [1].

Acknowledgements to Felix Joachimski and to the anonymous referee for their valuable remarks on preliminary versions of this text.

References

[1] Thorsten Altenkirch. Logical relations and inductive/coinductive types. In Georg Gottlob, Etienne Grandjean, and Katrin Seyr, editors, *Computer Science Logic, 12th International Workshop, Brno, Czech Republic, August 24–28, 1998, Proceedings*, volume 1584 of *Lecture Notes in Computer Science*, pages 343–354. Springer Verlag, 1999.

[2] Henk P. Barendregt. Lambda calculi with types. In Samson Abramsky, Dov M. Gabbay, and Tom S. E. Maibaum, editors, *Background: Computational Structures*, volume 2 of *Handbook of Logic in Computer Science*, pages 117–309. Oxford University Press, 1993.

[3] Herman Geuvers. Inductive and coinductive types with iteration and recursion. In Bengt Nordström, Kent Pettersson, and Gordon Plotkin, editors, *Proceedings of the 1992 Workshop on Types for Proofs and Programs, Båstad, Sweden, June 1992*, pages 193–217, 1992. Only published via `ftp://ftp.cs.chalmers.se/pub/cs-reports/baastad.92/proc.dvi.Z`.

[4] Jean-Yves Girard. *Interprétation fonctionnelle et élimination des coupures dans l'arithmétique d'ordre supérieur.* Thèse de Doctorat d'État, Université de Paris VII, 1972.

[5] Jean-Yves Girard, Yves Lafont, and Paul Taylor. *Proofs and Types*, volume 7 of *Cambridge Tracts in Theoretical Computer Science.* Cambridge University Press, 1989.

[6] Daniel Leivant. Contracting proofs to programs. In Piergiorgio Odifreddi, editor, *Logic and Computer Science*, volume 31 of *APIC Studies in Data Processing*, pages 279–327. Academic Press, 1990.

[7] Ralph Matthes. *Extensions of System* F *by Iteration and Primitive Recursion on Monotone Inductive Types.* Doktorarbeit (PhD thesis), University of Munich, 1998. Available via the homepage `http://www.tcs.informatik.uni-muenchen.de/~matthes/`.

[8] Ralph Matthes. Monotone fixed-point types and strong normalization. In Georg Gottlob, Etienne Grandjean, and Katrin Seyr, editors, *Computer Science Logic, 12th International Workshop, Brno, Czech Republic, August 24–28, 1998, Proceedings*, volume 1584 of *Lecture Notes in Computer Science*, pages 298–312. Springer Verlag, 1999.

[9] Ralph Matthes and Felix Joachimski. Short proofs of normalization for the simply-typed lambda-calculus, permutative conversions and Gödel's T. Submitted to the Archive for Mathematical Logic, 1998.

[10] John C. Reynolds. Towards a theory of type structure. In B. Robinet, editor, *Programming Symposium*, volume 19 of *Lecture Notes in Computer Science*, pages 408–425, Berlin, 1974. Springer-Verlag.

[11] Masako Takahashi. Parallel reduction in λ-calculus. *Information and Computation*, 118(1):120–127, 1995.

Communicated by (The editor will be set by the publisher).
June 2, 1999.