Inductive and Coinductive types with Iteration and Recursion in a Polymorphic Framework

Herman Geuvers,
Faculty of Mathematics and Computer Science,
University of Nijmegen,
Toernooiveld 1,
6525 ED Nijmegen,
The Netherlands

February 1992

Abstract

We study (extensions of) polymorphic typed lambda calculus from a point of view of how iterative and recursive functions on inductive types are represented. The inductive types can usually be understood as initial algebras in a certain category and then recursion can be defined in terms of iteration. However, in the syntax we often have only weak initiality, which makes the definition of recursion in terms of iteration inefficient or just impossible. We propose a categorical notion of (primitive) recursion which can easily be added as computation rule to a typed lambda calculus and gives us a clear view on what the dual of recursion, corecursion, on coinductive types is. (The same notion has, independently, been proposed by [Mendler 1991].) We then look at how these syntactic notions work out in the framework of K-models for polymorphic lambda calculus. It will turn out that with some quite weak extra assumptions, recursion can be defined in terms of corecursion and vice versa using polymorphism. This also works syntactically: We shall look at some slight extensions of polymorphic lambda calculus for which a scheme for either recursion or corecursion suffices to be able to define the other. As an application of this we look at the Calculus of Inductive Definitions ([Coquand and Mohring 1990] and [Dowek e.a. 1991]), which reflects our categorical notion of recursion and we show how to define coinductive types with corecursion in it.

1 Introduction

In this paper we want to look at formalizations of inductive and coinductive types in different typed lambda calculi, mainly extensions of the polymorphic lambda calculus. It is well-known that in polymorphic lambda calculus, many inductive data types can be defined (see e.g.[Böhm and Berarducci 1985] and [Girard et al. 1989]). In this paper we want to look at *how* functions on inductive types can be represented. Therefore, two ways of using the inductive building up of a type to define functions on that type are being distinguished, the *iterative* way and the *recursive* way. An *iterative* function is defined by induction on the building up of the type by defining the function value in terms of the previous values. A *recursive* function is also defined by induction, but now by defining the function value in terms of the previous values and the previous inputs. For functions on the natural numbers that is $h : \mathrm{Nat} \to A$, with $h(0) = c, h(n + 1) = f(h(n))$ (for $c : A, f : A \to A$) is iterative and $h : \mathrm{Nat} \to A$, with $h(0) = c, h(n + 1) = g(h(n), n)$ (for $c : A, g : A \times \mathrm{Nat} \to A$) is recursive. If one has pairing, the recursive functions can be defined using just iteration, which was essentially already shown by [Kleene 1936]. But if we work in a typed lambda calculus where pairing is not surjective, this translation of recursion in terms of iteration becomes inefficient and sometimes impossible. Moreover, if the calculus also incorporates some predicate logic, one would like to use the inductivity in doing proofs, which is not always straightforward (or just impossible.)

This asks for an explicit scheme for recursion in typed lambda calculus, which yields for, say, the natural numbers the scheme of Gödels T and (if we have predicate logic in the calculus) the induction priciple. To see how this can be done in general for inductive types, we are going to define a categorical notion of recursion (just like 'initial algebra' categorically represents the notion of iteration). One of the trade-offs is that we can dualize all this to get a notion of *corecursion* on coinductive types. These categorical notions of recursion and corecursion have independently been found by Mendler (see [Mendler 1991]) who treats these constructions in Martin-Löf type theory with predicative universes. What we define as (co)recursive (co)algebras are what Mendler calls '(co)algebras that admit simple primitive recursion'. We shall always use the term 'recursion', because, although the function-definition-scheme has a strong flavour of primitive recursion, one can define a lot more functions in polymorphic lambda calculus then just the primitive recursive ones. Coinductive types were first described in [Hagino 1987a] and [Hagino 1987b], with only a scheme for coiteration and without corecursion.

A very surprising result will be that in a polymorphic framework, if we have a notion of recursive types which reflects our notion of recursive algebra, then we can define corecursive types that correspond to corecursive coalgebras. By duality, this also works the other way around. This result will be given semantically and syntactically: For the semantics we look at a slight extension of K*-models* (defined in [Reynolds and Plotkin 1990]), which is a very general notion of model for polymorphic lambda calculus, covering most of the known models. We shall show that if we have recursive T-algebras for every functor T that is expressible in the syntax, then we have corecursive T-coalgebras for every functor T that is expressible in the syntax. For the syntax we look at a system of recursive and corecursive types defined by [Mendler 1987] and show that with either the scheme for recursive types or the scheme for corecursive types, there is a recursive Φ-algebra and a corecursive Φ-coalgebra in the syntax for every syntactic functor Φ (where syntactic functors are positive type schemes.) We also look at a syntax with recursive and corecursive types that straightforwardly represents the categorical notions of recursive algebra and corecursive coalgebra; for this system it will be shown that either one of the schemes suffices to obtain the other.

Finally we shall treat the Calculus of Inductive Definitions, as described by [Coquand and Mohring 1990] and implemented as 'Coq' by [Dowek e.a. 1991]. This calculus has a scheme for defining inductive types and we shall show that also coinductive types can be defined with this scheme.

2 The categorical perspective

As said, we shall get our intuitions about inductive and coinductive types from the field of category theory. The main notions in category theory related to this issue come from [Lambek 1968].

Definition 2.1 *Let C be a category, T a functor from C to C.*

1. *A T*-algebra *in C is a pair (A, f), with A an object and $f : TA \to A$.*
2. *If (A, f) and (B, g) are T-algebra's,* a morphism from (A, f) to (B, g) *is a morphism $h : A \to B$ such that the following diagram commutes.*

$$\begin{tikzcd} TA \arrow[r, "f"] \arrow[d, "Th"'] \arrow[dr, phantom, "="] & A \arrow[d, "h"] \\ TB \arrow[r, "g"'] & B \end{tikzcd}$$

3. *A T-algebra (A, f) is* initial *if it is initial in the category of T-algebras, i.e. for every T-algebra (B, g) there's a unique h which makes the above diagram commute.*

In a category with products, coproducts and terminal object, the initial algebra of the functor $TX = 1 + X$ is the natural numbers object, for which we write $(\mathsf{Nat}, [\mathsf{Z}, \mathsf{S}])$. The initial algebra of $TX = 1 + (A \times X)$ is the object of finite lists over A, $(\mathsf{List}_A, [\mathsf{Nil}, \mathsf{Cons}])$. In this paper our pet-example of an initial algebra will be $(\mathsf{Nat}, [\mathsf{Z}, \mathsf{S}])$, which will be used to illustrate the properties we are interested in. First take a look at how the iterative and recursive functions can be defined on Nat. (The example immediately generalizes to arbitrary initial algebras.)

Example 2.2 *1. For $g : 1 + B \to B$ we write g_1 for $g \circ \mathsf{in}_1 : 1 \to B$ and g_2 for $g \circ \mathsf{in}_2 : B \to B$. The* iteratively defined morphism from g_1, g_2, $\mathsf{Elim}g_1g_2$, *is defined as the unique morphism h which makes the diagram commute, i.e. $h \circ \mathsf{Z} = g_1$ and $h \circ \mathsf{S} = g_2 \circ h$.*

2. For $g_1 : 1 \to B$, $g_2 : B \times \mathsf{Nat} \to B$, the recursively defined morphism from g_1 *and* g_2 *is constructed as follows.*
There exists a unique h which makes the diagram

$$\begin{tikzcd}
1 + \mathsf{Nat} \arrow[r, "{[\mathsf{Z}, \mathsf{S}]}"] \arrow[d, "\mathsf{id} + h"'] \arrow[dr, phantom, "="] & \mathsf{Nat} \arrow[d, "h"] \\
1 + (B \times \mathsf{Nat}) \arrow[r, "{\langle [g_1, g_2], [\mathsf{Z}, \mathsf{S} \circ \pi_2] \rangle}"'] & B \times \mathsf{Nat}
\end{tikzcd}$$

commute. That is $h \circ [\mathsf{Z}, \mathsf{S}] = \langle [g_1, g_2], [\mathsf{Z}, \mathsf{S} \circ \pi_2] \rangle \circ \mathsf{id} + h$. If we write $h_1 = \pi_1 \circ h$ and $h_2 = \pi_2 \circ h$ we have the equalities

$$\begin{aligned}
h_1 \circ \mathsf{Z} &= g_1, \\
h_1 \circ \mathsf{S} &= g_2 \circ h, \\
h_2 \circ \mathsf{Z} &= \mathsf{Z}, \\
h_2 \circ \mathsf{S} &= \mathsf{S} \circ h_2.
\end{aligned}$$

Now $h_2 = \mathsf{id}_{Nat}$ by uniqueness and also $h = \langle h_1, h_2 \rangle$, so

$$\begin{aligned}
h_1 \circ \mathsf{Z} &= g_1, \\
h_1 \circ \mathsf{S} &= g_2 \circ \langle h_1, \mathsf{id} \rangle.
\end{aligned}$$

So h_1 satisfies the recursion equalities and we define

$$\mathsf{Rec}g_1g_2 := h_1.$$

Definition 2.3 *Let C be a category, T a functor from C to C.*

1. *A T-*coalgebra in C *is a pair (A, f), with A an object and $f : A \to TA$.*

2. *If (A, f) and (B, g) are T-coalgebras,* a morphism from (B, g) to (A, f) *is a morphism $h : B \to A$ such that the following diagram commutes.*

$$\begin{tikzcd}
B \arrow[r, "g"] \arrow[d, "h"'] \arrow[dr, phantom, "="] & TB \arrow[d, "Th"] \\
A \arrow[r, "f"'] & TA
\end{tikzcd}$$

3. *A T-coalgebra (A, f) is* terminal *if it is terminal in the category of T-coalgebras, i.e. for every coalgebra (B, g) there's a unique h which makes the above diagram commute.*

Our pet example for terminal coalgebras is the one for $TX = \mathsf{Nat} \times X$, the object of inifinite lists of natural numbers, for which we write $(\mathsf{Stream}, \langle \mathsf{H}, \mathsf{T} \rangle)$. We shall dualize the notions of iterative and recursive function to get *coiterative* and *corecursive* functions *to* Stream. (Again this example easily generalizes to the case for arbitrary terminal coalgebras.)

Example 2.4 1. *For $g : B \to \mathsf{Nat} \times B$, write g_1 for $\pi_1 \circ g : B \to \mathsf{Nat}$ and g_2 for $\pi_2 \circ g : B \to B$. The coiteratively defined morphism from g_1 and g_2, $\mathsf{Intro} g_1 g_2 : B \to \mathsf{Stream}$ is the (unique) morphism h for which the diagram commutes. That is, $H \circ h = g_1$ and $T \circ h = h \circ g_2$.*
If j is a morphism from Nat *to* Nat*, one can define the morphism from* Stream *to* Stream *which applies f to every point in the stream as $\mathsf{Intro}(j \circ \mathsf{H})T$. Note that it is not so straightforward to define (coiteratively) a morphism which replaces the head of a stream by, say, zero. This, however, can easily be done using corecursion.*

2. *For $g_1 : B \to \mathsf{Nat}$, $g_2 : B \to B + \mathsf{Stream}$, the corecursively defined morphism from g_1 and g_2, $\mathsf{Corec} g_1 g_2$ is defined by $h \circ \mathsf{in}_1$, where h is the (unique) morphism which makes the diagram*

$$\begin{tikzcd}
B + \mathsf{Stream} \arrow[r, "{[\langle g_1, g_2\rangle, \langle \mathsf{H}, \mathsf{in}_2 \circ \mathsf{T}\rangle]}"] \arrow[d, "h"'] \arrow[dr, phantom, "="] & \mathsf{Nat} \times (B + \mathsf{Stream}) \arrow[d, "\mathsf{id} \times h"] \\
\mathsf{Stream} \arrow[r, "{\langle \mathsf{H}, \mathsf{T} \rangle}"'] & \mathsf{Nat} \times \mathsf{Stream}
\end{tikzcd}$$

commute. If we write $h_1 = h \circ \mathsf{in}_1, h_2 = h \circ \mathsf{in}_2$, then we have for h the following equations

$$\begin{aligned}
\mathsf{H} \circ h_2 &= \mathsf{H}, \\
\mathsf{T} \circ h_2 &= h_2 \circ \mathsf{T}, \\
\mathsf{H} \circ h_1 &= g_1, \\
\mathsf{T} \circ h_1 &= h \circ g_2.
\end{aligned}$$

Now $h_2 = \mathsf{id}$ by uniqueness and also $h = [h_1, h_2]$, so

$$\begin{aligned}
\mathsf{H} \circ h_1 &= g_1, \\
\mathsf{T} \circ h_1 &= [h_1, \mathsf{id}] \circ g_2.
\end{aligned}$$

These are the equations for corecursion; if $g_1 : B \to \mathsf{Nat}$ and $g_2 : B \to B + \mathsf{Stream}$, then $j : B \to \mathsf{Stream}$ is corecursively defined from g_1 and g_2 *if $\mathsf{H} \circ j = g_1$ and $\mathsf{T} \circ j = [j, \mathsf{id}] \circ g_2$.*
The function $\mathsf{ZeroH} : \mathsf{Stream} \to \mathsf{Stream}$ which changes the head of a stream into zero can now be defined as $\mathsf{ZeroH} := \mathsf{Corec}(\mathsf{Z}\circ!)(\mathsf{in}_2 \circ \mathsf{T})$, where $!$ is the unique morphism from Stream *to* 1. *(Informally, $\mathsf{Z}\circ!$ is of course just $\lambda s : \mathsf{Stream}.0$.)*

As usual in categorical definitions, the definitions of initial algebra and terminal coalgebra split up in two parts, the 'existence part' (there's an h such that...) and the 'uniqueness part' (the h is unique.) In the following we shall sometimes refer to these two parts of the definition as the *existence property* and the *uniqueness property*.

In the typed lambda calculi that we shall consider, the inductive and coinductive types will not exactly represent initial algebras and terminal coalgebras. What the systems are lacking is the uniqueness property for the morphism h in 2.1, respectively 2.3. Algebras, respectively coalgebras, which only satisfy the existence property are called *weakly initial*, respectively *weakly terminal*.

Definition 2.5 *For T an endofunctor in a category C, The T-algebra, respectively T-coalgebra, (A, f) is* weakly initial, *respectively* weakly terminal, *if for every T-algebra, respectively T-coalgebra, (B, g) there exists an arrow h that makes the diagram in 2.1, respectively 2.3, commute.*

Remark 2.6 *The notion of weakly initial algebra is really weaker than that of initial algebra. This can easily be seen by noticing that in the category* **Set**, *$(2\omega, [\mathsf{Z}, \mathsf{S}])$ is a weakly initial $(\lambda X.1 + X)$-algebra, but also $(2\omega, [\mathsf{Z}, \mathsf{S}'])$, with $\mathsf{S}'(n) = \mathsf{S}(n)$, $\mathsf{S}'(\omega + n) = n$ is. (On weakly initial algebras, the behaviour of morphisms is only determined on the standard part of the algebra, that is in seththeoretic terms, those elements that are constructed by finitely many times applying the constructor f. Real initiality says that the algebra is standard.)*

As we made serious use of the uniqueness property in constructing the recursive and corecursive functions, it's interesting to see how much we can do in weak initial algebras and weak terminal coalgebras. The construction of the iterative and coiterative functions of examples 2.2 and 2.4 can be done in the same way; we only loose the uniqueness property of the iteratively defined function. The construction of recursive and corecursive functions in a weak framework is not so straightforward. We shall study again the examples of natural numbers and streams of natural numbers. Fix a category C, which has weak products and coproducts. (So we *do* have e.g. $\pi_1 \circ \langle t_1, t_2 \rangle = t_1$ and $[t_1, t_2] \circ \mathsf{in}_1 = t_1$, but not $\langle \pi_1 \circ t, \pi_2 \circ t \rangle = t$ and $[t \circ \mathsf{in}_1, t \circ \mathsf{in}_2] = t$.) It will turn out that weak products and coproducts will cause some extra restrictions on the definability of functions. Therefore we shall also study what happens if product and coproduct are *semi*, that is for products $\langle f, g \rangle \circ h = \langle f \circ h, g \circ h \rangle$ and for coproducts $h \circ [f, g] = [h \circ f, h \circ g]$. The reason for not considering the strong products and coproducts in these examples is that in the syntax of typed lambda calculi product and coproduct are usually weak or semi. (The notions of semi product and semi coproduct are taken from [Hayashi 1985].)

Example 2.7 *(Recursion on a weak natural numbers object) Let* Nat *be a weakly initial $\lambda X.1+X$-algebra. Consider the diagram in 2.2, where we defined recursion in terms of iteration and let $h : \mathsf{Nat} \to B \times \mathsf{Nat}$ be some morphism that makes the diagram commute. Then also $\langle \pi_1 \circ h, \pi_2 \circ h \rangle$ makes the diagram commute. Write $h_1 = \pi_1 \circ h$ and $h_2 = \pi_2 \circ h$. We have the following equalities.*

$$\begin{aligned} h_1 \circ \mathsf{Z} &= g_1, \\ h_1 \circ \mathsf{S} &= g_2 \circ h, \\ h_2 \circ \mathsf{Z} &= \mathsf{Z}, \\ h_2 \circ \mathsf{S} &= \mathsf{S} \circ h_2. \end{aligned}$$

Nat *doesn't satisfy the uniqueness properties, so not necessarily $h_2 = \mathsf{id}_{Nat}$but only*

$$h_2 \circ \mathsf{S}^n \circ \mathsf{Z} = \mathsf{S}^n \circ \mathsf{Z}$$

for every $n \in \mathbf{N}$, where S^n denotes an n-fold composition of S. Now we would like to deduce

$$\begin{aligned} h_1 \circ \mathsf{Z} &= g_1, \\ h_1 \circ \mathsf{S}^{n+1} \circ \mathsf{Z} &= g_2 \circ \langle h_1, \mathsf{id} \rangle \circ \mathsf{S}^n \circ \mathsf{Z}, \end{aligned}$$

which says that h_1 satisfies the recusion equations for the 'standard' natural numbers.

- *For weak products this conclusion is only valid if $g_2 = k \circ \pi_i$ for some $k : B \to B$ or $k : \mathsf{Nat} \to B$. (Note that if $g_2 = k \circ \pi_1$ for some $k : B \to B$, then h_1 is just iteratively defined from g_1 and k, so only the case for $g_2 = k \circ \pi_2$ gives us really new functions, for instance the predecessor.)*

- *For semi products this conclusion is only valid for $g_2 = k \circ \langle \pi_1, \pi_2 \rangle$ for some $k : B \times \mathsf{Nat} \to B$, which is not a serious restriction: Just replace g_2 by $g_2 \circ \langle \pi_1, \pi_2 \rangle$.*

Example 2.8 *(Corecursion on a weak stream object) Let* Stream *be a weakly terminal $\lambda X.\mathsf{Nat} \times X$-coalgebra. Consider the diagram in 2.4, where we defined corecursion in terms of coiteration and let*

$h : (B + \mathsf{Stream}) \to \mathsf{Stream}$ be some morphism that makes the diagram commute. Write $h_1 = h \circ \mathsf{in}_1$ and $h_2 = h \circ \mathsf{in}_2$. We have the following equalities.

$$\begin{aligned} \mathsf{H} \circ h_2 &= \mathsf{H}, \\ \mathsf{T} \circ h_2 &= h_2 \circ \mathsf{T}, \\ \mathsf{H} \circ h_1 &= g_1, \\ \mathsf{T} \circ h_1 &= h \circ g_2. \end{aligned}$$

Now we can not conclude $h \circ \mathsf{in}_2 = \mathsf{id}$, because we don't have uniqueness, but we do have

$$\mathsf{H} \circ \mathsf{T}^n \circ h_2 = \mathsf{H} \circ \mathsf{T}^n,$$

that is h_2 is the identity on the 'standard' part of the stream (those points that can be obtained by finitely many applications of H or T.) Again we would like to conclude

$$\begin{aligned} \mathsf{H} \circ h_1 &= g_1, \\ \mathsf{H} \circ \mathsf{T}^{n+1} \circ h_1 &= \mathsf{H} \circ \mathsf{T}^n \circ \big[h_1, \mathrm{id}\big] \circ g_2, \end{aligned}$$

that is h_1 satisfies the corecursion equations for the 'standard' part of the stream.

- *For weak coproducts this conclusion is only valid if $g_2 = \mathsf{in}_i \circ k$ for some $k : B \to B$ or $k : B \to \mathsf{Stream}$. (Note that if $g_2 = \mathsf{in}_i \circ k$ for some $k : B \to B$, then h_1 is just coiteratively defined from g_1 and k, so only the case for $g_2 = \mathsf{in}_2 \circ k$ gives us really new functions, like for instance the function ZeroH.)*

- *For semi coproducts this conclusion is only valid if $g_2 = \big[\mathsf{in}_1, \mathsf{in}_2\big] \circ k$ for some $k : B \to B + \mathsf{Stream}$. again this is not a serious restriction: Just replace g_2 by $\big[\mathsf{in}_1, \mathsf{in}_2\big] \circ k$.*

For the morphism $\mathsf{ZeroH} : \mathsf{Stream} \to \mathsf{Stream}$ which replaces the head by zero, defined in 2.4 by $\mathsf{Corec}(\mathsf{Z}\circ !)(\mathsf{in}_2 \circ \mathsf{T})$, we now have (for either weak or semi coproducts)

$$\begin{aligned} \mathsf{H} \circ \mathsf{ZeroH} &= \mathsf{Z}, \\ \mathsf{H} \circ \mathsf{T}^{n+1} \circ \mathsf{ZeroH} &= \mathsf{H} \circ \mathsf{T}^{n+1}, \end{aligned}$$

so ZeroH works fine on the standard part of the stream. That one can not, in general, define a morphism ZeroH such that $\mathsf{T} \circ \mathsf{ZeroH} = \mathsf{T}$ will be shown later, when we look at these examples inpolymorphic lambda calculus which is an instance of a category with weakly initial algebras and weakly terminal coalgebras, semi products and weak coproducts.

Remark 2.9 *With strong products and coproducts we would have similar problems in defining recursion and corecursion. The recursion equations would only be valid for the standard natural numbers and the corecursion equations would only be valid for the standard part of streams. The only advantage would be that the $g_2 : B \times \mathsf{Nat} \to B$, respectively the $g_2 : B \to B + \mathsf{Stream}$ can be taken arbitrarily.*

In section 3 the polymorphic lambda calculus will be considered in which inductive and coinductive types can be defined which correspond to weakly initial algebras and weakly terminal coalgebras. It will be shown that recursion in that calculus is problematic from a point of view of efficiency. One solution could be to strengthen the reduction rules to get a stronger (extensional) equality. However, it's not possible to add some relatively easy reduction rules to the syntax to obtain the uniqueness property of initiality and terminality. (We can't say in an easy way that the only objects of a structure are the standard ones.) This is because the equality of (primitive) recursive functions can not be decided by an easy (decidable) equality. We can do something different, namely say that our functions should behave on the non-standard part as they behave on the standard part. Categorically, this can be obtained by strengthening the notion of weakly initial algebra and weakly terminal coalgebra a little bit, such that recursion 'works'. (That is for $\mathbf{N}$, for $c : A, g : A \times \mathrm{Nat} \to A$ there is a function $h : \mathrm{Nat} \to A$, with $h(0) = c$ and $h(n+1) = g(h(n), n)$.) These new notions will be called *recursive algebra* and *corecursive coalgebra*.

The definitions are not difficult if one understands what makes it possible to define (co)recursion, in terms of (co)iteration.

Let in the following C be a category with weak products and weak coproducts and T a functor from C to C.

Definition 2.10 *(A,f) is a* recursive T-algebra *if (A,f) is a T-algebra and for every $g : T(X \times A) \to X$ there exists an $h : A \to X$ such that the following diagram commutes.*

$$\begin{tikzcd}
TA \arrow[r, "f"] \arrow[d, "T(\langle h, \mathsf{id}\rangle)"'] \arrow[dr, phantom, "="] & A \arrow[d, "h"] \\
T(X \times A) \arrow[r, "g"'] & X
\end{tikzcd}$$

Notice that this is the same as saying that (A,f) is weakly initial and that moreover, in the diagram for defining recursion in terms of iteration, $h_2 = \mathsf{id}$. (See 2.2)

Definition 2.11 *(A,f) is a* corecursive T-coalgebra *if (A,f) is a T-coalgebra and for every $g : X \to T(X + A)$ there exists an $h : X \to A$ such that the following diagram commutes.*

$$\begin{tikzcd}
X \arrow[r, "g"] \arrow[d, "h"'] \arrow[dr, phantom, "="] & T(X + A) \arrow[d, "T(\lbrack h, \mathsf{id}\rbrack)"] \\
A \arrow[r, "f"'] & TA
\end{tikzcd}$$

Again this is the same as saying that (A,f) is a weakly terminal T-coalgebra and that moreover, in the diagram for defining corecursion in terms of coiteration, $h_2 = \mathsf{id}$. (See 2.4)

When talking about weakly initial or recursive T-algebras and weakly terminal or corecursive T-coalgebras, it is convenient to denote the h that makes the diagram commute as a function of g. So we shall denote a weakly initial T-algebra by (A, f, Elim), where $\mathsf{Elim}g$ denotes a morphism h in 2.5 that makes the diagram commute. Similarly, we write (A, f, Intro) for a weakly terminal T-coalgebra, (A, f, Rec) for a recursive T-algebra and (A, f, Corec) for a corecursive T-coalgebra.

Examples 2.12

1. *If $(\mathsf{Nat}, [\mathsf{Z}, \mathsf{S}], \mathsf{Rec})$ is a recursive $\lambda X.1 + X$-algebra, Rec is a recursor on Nat: For $[g_1, g_2] : 1 + (X \times \mathsf{Nat}) \to X$,*

$$\begin{aligned}
\mathsf{Rec}[g_1, g_2] \circ \mathsf{Z} &= g_1, \\
\mathsf{Rec}[g_1, g_2] \circ \mathsf{S} &= g_2 \circ \langle \mathsf{Rec}[g_1, g_2], \mathsf{id}\rangle,
\end{aligned}$$

so $\mathsf{Rec}[g_1, g_2]$ is the recursively defined function from g_1 and g_2. We can define $\mathsf{P} := \mathsf{Rec}[\mathsf{Z}, \pi_2]$ and we have

$$\begin{aligned}
\mathsf{P} \circ \mathsf{Z} &= \mathsf{Z}, \\
\mathsf{P} \circ \mathsf{S} &= \mathsf{id}.
\end{aligned}$$

2. *If $(\mathsf{Stream}, \langle \mathsf{H}, \mathsf{T}\rangle, \mathsf{Corec})$ is a corecursive $\lambda X.\mathsf{Nat} \times X$-coalgebra. Then for $\langle g_1, g_2\rangle : X \to \mathsf{Nat} \times (X + \mathsf{Stream})$, the function $\mathsf{Corec}\langle g_1, g_2\rangle$ satisfies*

$$\begin{aligned}
H \circ \mathsf{Corec}\langle g_1, g_2\rangle &= g_1, \\
T \circ \mathsf{Corec}\langle g_1, g_2\rangle &= [\mathsf{Corec}\langle g_1, g_2\rangle, \mathsf{id}] \circ g_2,
\end{aligned}$$

so $\mathsf{Corec}\langle g_1, g_2\rangle$ *is the corecursively defined function from* g_1 *and* g_2. *We can define*

$$\mathsf{ZeroH} := \mathsf{Corec}\langle \mathsf{Z}\circ !, \mathsf{in}_2 \circ \mathsf{T}\rangle$$

with

$$\begin{aligned} \mathsf{H} \circ \mathsf{ZeroH} &= \mathsf{Z}\circ !, \\ \mathsf{T} \circ \mathsf{ZeroH} &= \mathsf{T}. \end{aligned}$$

3 The syntax of polymorphic lambda calculus

We just give the rules to fix our notation and shall not go into the system further, assuming it is familiar. For convenience we extend the syntax of polymorphic lambda calculus with weak products and coproducts. (This is of course a conservative extension, because weak products and coproducts are definable: $\sigma \times \tau \equiv \forall\alpha.(\sigma\to\tau\to\alpha)\to\alpha$ and $\sigma + \tau \equiv \forall\alpha.(\sigma\to\alpha)\to(\tau\to\alpha)\to\alpha$.) We shall call the system $F^{\times+}$, to stress the fact that it is just system F (the polymorphic lambda calculus) as defined in [Girard et al. 1989]), [Girard 1972] or [Reynolds 1974]) with explicit product and coproduct constructors.

Definition 3.1 *1. The set of types of* $F^{\times+}$, $\mathbf{T}$, *is defined by the following abstract syntax.*

$$\mathbf{T} ::= \mathit{TypVar} \,|\, \mathbf{T}\to\mathbf{T} \,|\, \mathbf{T}\times\mathbf{T} \,|\, \mathbf{T}+\mathbf{T} \,|\, \forall \mathit{TypVar}.\mathbf{T}$$

2. The expressions of $F^{\times+}$, T, *are defined by the following abstract syntax.*

$$T ::= \mathit{Var} \,|\, \mathrm{fst}^{\mathbf{TT}} \,|\, \mathrm{snd}^{\mathbf{TT}} \,|\, \mathrm{inl}^{\mathbf{TT}} \,|\, \mathrm{inr}^{\mathbf{TT}} \,|\, TT \,|\, {<}T,T{>} \,|\, [T,T] \,|\, T\mathbf{T} \,|\lambda\, \mathit{Var}{:}\mathbf{T}.T \,|\, \Lambda \mathit{TypVar}.T$$

3. A context *is a sequence of* declarations $x{:}\sigma$ *(*$x \in \mathit{Var}$ *and* $\sigma \in \mathbf{T}$*), where it is assumed that if* $x{:}\sigma$ *and* $y{:}\tau$ *are different declarations in the same context, then* $x \not\equiv y$.

4. The typing rules for deriving judgements of the form $\Gamma \vdash M{:}\sigma$ *for* Γ *a context,* M *an expression and* σ *a type, are the following.*

- *If* $x{:}\sigma$ *is in* Γ, *then* $\Gamma \vdash x{:}\sigma$,
- $\Gamma \vdash \mathrm{fst}^{\sigma\tau}{:}\sigma\times\tau\to\sigma$ *and* $\Gamma \vdash \mathrm{snd}^{\sigma\tau}{:}\sigma\times\tau\to\tau$,
- $\Gamma \vdash \mathrm{inl}^{\sigma\tau}{:}\sigma\to\sigma+\tau$ *and* $\Gamma \vdash \mathrm{inr}^{\sigma\tau}{:}\tau\to\sigma+\tau$,
- $\dfrac{\Gamma \vdash M{:}\sigma\to\tau \quad \Gamma \vdash N{:}\sigma}{\Gamma \vdash MN{:}\tau} \qquad \dfrac{\Gamma, x{:}\sigma \vdash M{:}\tau}{\Gamma \vdash \lambda x{:}\sigma.M{:}\sigma\to\tau}$
- $\dfrac{\Gamma \vdash M{:}\forall\alpha.\sigma}{\Gamma \vdash M\tau{:}\sigma[\tau/\alpha]}$ *if* $\tau \in \mathbf{T}$. $\qquad \dfrac{\Gamma \vdash M{:}\sigma}{\Gamma \vdash \Lambda\alpha.M{:}\forall\alpha.\sigma}$ *if* $\alpha \notin FTV(\Gamma)$.
- $\dfrac{\Gamma \vdash M{:}\sigma\to\tau \quad \Gamma \vdash N{:}\sigma\to\rho}{\Gamma \vdash {<}M,N{>}{:}\sigma\to\tau\times\rho} \qquad \dfrac{\Gamma \vdash M{:}\tau\to\sigma \quad \Gamma \vdash N{:}\rho\to\sigma}{\Gamma \vdash [M,N]{:}\tau+\rho\to\sigma}$

FTV denotes the set of free type variables (TypVar.)

5. The one step reduction rules are the following.

- $(\lambda x{:}\sigma.M)N \longrightarrow_\beta M[N/x]$,
- $\lambda x{:}\sigma.Mx \longrightarrow_\eta M$ *if* $x \notin FV(M)$,
- $(\Lambda\alpha.M)\tau \longrightarrow_\beta M[\tau/\alpha]$,
- $\mathrm{fst}^{\sigma\tau} \circ {<}f_1, f_2{>} \longrightarrow_\times f_1$, *and* $\mathrm{snd}^{\sigma\tau} \circ {<}f_1, f_2{>} \longrightarrow_\times f_2$,
- $[f_1, f_2] \circ \mathrm{inl}^{\sigma\tau} \longrightarrow_+ f_1$, *and* $[f_1, f_2] \circ \mathrm{inr}^{\sigma\tau} \longrightarrow_+ f_2$.

FV denotes the free term variables (Var.) One step reduction, $\longrightarrow$, is defined as the union of $\longrightarrow_\beta$, $\longrightarrow_\eta$, $\longrightarrow_\beta$, $\longrightarrow_\times$ and $\longrightarrow_+$. The relations $\twoheadrightarrow$ and $=$ are respectively defined as the transitive, reflexive and the transitive, reflexive, symmetric closure of $\longrightarrow$.

Here, $t'[t/u]$ denotes the substitution of t for the variable u in t'. Substitution is done with the usual care, renaming bound variables such that no free variable becomes bound after substitution.

Type variables will be denoted by the lower case Greek characters α, β and γ, term variables will be denoted by lower case Roman characters. If there is no ambiguity, the superscripts to fst, snd, inl and inr will be omitted. The set of expressions typable in the context Γ with type σ is denoted by $\mathrm{Term}(\sigma, \Gamma)$.

We want to discuss categorical notions like weak initiality in the syntax and therefore define need a syntactic notion of functor. This will be covered by the (well-known) notion of positive or negative type scheme.

Definition 3.2 *1. A* type scheme *in $F^{\times+}$ is a type $\Phi(\alpha)$ where α marks all occurrences (possibly none) of α.*

2. A type scheme $\Phi(\alpha)$ can be positive *or* negative *(but also none of the both), which is defined by induction on the structure of $\Phi(\alpha)$ as follows.*

(a) If $\alpha \notin FTV(\Phi(\alpha))$, then $\Phi(\alpha)$ is positive *and* negative,

(b) if $\Phi(\alpha) \equiv \alpha$ then $\Phi(\alpha)$ is positive,

(c) if $\Phi(\alpha) \equiv \Phi_1(\alpha){\to}\Phi_2(\alpha)$, $\Phi_1(\alpha)$ is negative and $\Phi_2(\alpha)$ is positive, then $\Phi(\alpha) \equiv \Phi_1(\alpha){\to}\Phi_2(\alpha)$ is positive,

(d) if $\Phi(\alpha) \equiv \Phi_1(\alpha){\to}\Phi_2(\alpha)$, $\Phi_1(\alpha)$ is positive and $\Phi_2(\alpha)$ is negative, then $\Phi(\alpha) \equiv \Phi_1(\alpha){\to}\Phi_2(\alpha)$ is negative,

(e) if $\Phi(\alpha) \equiv \forall\beta.\Phi'(\alpha)$ and $\Phi'(\alpha)$ is positive (resp. negative) then $\Phi(\alpha) \equiv \forall\beta.\Phi'(\alpha)$ is positive *(resp.* negative*),*

(f) if $\Phi(\alpha) \equiv \Phi_1(\alpha) \times \Phi_2(\alpha)$ and $\Phi_1(\alpha)$ is positive (resp. negative) and $\Phi_2(\alpha)$ is positive (resp. negative), then $\Phi(\alpha) \equiv \Phi_1(\alpha) \times \Phi_2(\alpha)$ is positive *(resp.* negative*),*

(g) if $\Phi(\alpha) \equiv \Phi_1(\alpha) + \Phi_2(\alpha)$ and $\Phi_1(\alpha)$ is positive (resp. negative) and $\Phi_2(\alpha)$ is positive (resp. negative), then $\Phi(\alpha) \equiv \Phi_1(\alpha) + \Phi_2(\alpha)$ is positive *(resp.* negative*).*

3. A positive (resp. negative) type scheme $\Phi(\alpha)$ works covariantly (resp. contravariantly) on a term $f{:}\sigma{\to}\tau$, obtaining a term $\Phi(f)$ of type $\Phi(\sigma){\to}\Phi(\tau)$ (resp. $\Phi(\tau){\to}\Phi(\sigma)$), by lifting, *defined inductively as follows. (Let $f{:}\sigma{\to}\tau$.)*

(a) If $\alpha \notin FTV(\Phi(\alpha))$, then $\Phi(f) := \mathrm{id}_{\Phi(\alpha)}$,

(b) if $\Phi(\alpha) \equiv \alpha$ then $\Phi(f) := f$,

(c) if $\Phi(\alpha) \equiv \Phi_1(\alpha){\to}\Phi_2(\alpha)$ is positive, then $\Phi(f) := \lambda x{:}\Phi_1(\sigma){\to}\Phi_2(\sigma).\lambda y{:}\Phi_1(\tau).\Phi_2(f)(x(\Phi_1(f)y)))$,

(d) if $\Phi(\alpha) \equiv \Phi_1(\alpha){\to}\Phi_2(\alpha)$ is negative, then $\Phi(f) := \lambda x{:}\Phi_2(\tau){\to}\Phi_1(\tau).\lambda y{:}\Phi_2(\sigma).\Phi_1(f)(x(\Phi_2(f)y)))$,

(e) if $\Phi(\alpha) \equiv \forall\beta.\Phi'(\alpha)$ is positive, then $\Phi(f) := \lambda x{:}\Phi(\sigma).\lambda\beta.\Phi'(f)(x\beta)$,

(f) if $\Phi(\alpha) \equiv \forall\beta.\Phi'(\alpha)$ is negative, then $\Phi(f) := \lambda x{:}\Phi(\tau).\lambda\beta.\Phi'(f)(x\beta)$,

(g) if $\Phi(\alpha) \equiv \Phi_1(\alpha) \times \Phi_2(\alpha)$, then $\Phi(f) := \Phi_1(f) \times \Phi_2(f)$,

(h) if $\Phi(\alpha) \equiv \Phi_1(\alpha) + \Phi_2(\alpha)$, then $\Phi(f) := \Phi_1(f) + \Phi_2(f)$,

where, as usual, $g \times h := <g \circ \mathrm{fst}, h \circ \mathrm{snd}>$ and $g + h := [\mathrm{inl} \circ g, \mathrm{inr} \circ h]$.

It is easy to check that if a positive or negative type scheme does not contain $\times$ and $+$, then the lifting preserves identity and composition: $\Phi(\mathrm{id}) = \mathrm{id}$ and if $\Phi(\alpha)$ is positive then $\Phi(f \circ g) = \Phi(f) \circ \Phi(g)$, if $\Phi(\alpha)$ is negative then $\Phi(f \circ g) = \Phi(g) \circ \Phi(f)$. Positive type schemes without $\times$ and $+$ can really be viewed as covariant functors in the syntax of polymorphic lambda calculus and negative type schemes as contravariant functors. (Consider a syntax with countably many variables of every type and view the types as objects and the terms of type $\sigma{\to}\tau$ as morphisms from σ to τ.)

The positive type schemes are a syntactic version of covariant functors. Similarly we also have syntactic versions of weakly initial (terminal) (co)algebras and (co)recorsive (co)algebras.

Definition 3.3 *Suppose we work in (an extension of) polymorphic lambda calculus where we have fixed a notation for weak products and coproducts (e.g. the second order definable ones.) Let $\Phi(\alpha)$ be a positive type scheme.*

1. *The triple* $(\sigma_0, M_0, \mathrm{Elim})$ *is a* syntactic weakly initial Φ-algebra *if*

   (a) $\sigma_0 \in \mathbf{T}$,

   (b) $\vdash M_0{:}\Phi(\sigma_0)\to\sigma_0$,

   (c) $\vdash \mathrm{Elim}{:}\forall\beta.(\Phi(\beta)\to\beta)\to\sigma_0\to\beta$,

   such that
   $$\mathrm{Elim}\tau g \circ M_0 = g \circ \Phi(\mathrm{Elim}\tau g)$$
   for any $\tau \in \mathbf{T}$ *and* $\Gamma \vdash g{:}\Phi(\tau)\to\tau$.

2. *The triple* $(\sigma_1, M_1, \mathrm{Intro})$ *is a* syntactic weakly terminal Φ-coalgebra *if*

   (a) $\sigma_1 \in \mathbf{T}$,

   (b) $\vdash f_1{:}\sigma_1\to\Phi(\sigma_1)$,

   (c) $\vdash \mathrm{Intro}{:}\forall\beta.(\beta\to\Phi(\beta))\to\beta\to\sigma_1$,

   such that
   $$M_1 \circ \mathrm{Intro}\tau g = \Phi(\mathrm{Intro}\tau g) \circ g$$
   for any $\tau \in \mathbf{T}$ *and* $\Gamma \vdash g{:}\tau\to\Phi(\tau)$.

3. *The triple* $(\sigma_0, M_0, \mathrm{Rec})$ *is a* syntactic recursive Φ-algebra *if*

   (a) $\sigma_0 \in \mathbf{T}$,

   (b) $\vdash M_0{:}\Phi(\sigma_0)\to\sigma_0$,

   (c) $\vdash \mathrm{Rec}{:}\forall\beta.(\Phi(\beta\times\sigma_0)\to\beta)\to\sigma_0\to\beta$,

   such that
   $$\mathrm{Rec}\tau g \circ M_0 = g \circ \Phi(<\mathrm{Rec}\tau g, \mathrm{id}>)$$
   for any $\tau \in \mathbf{T}$ *and* $\Gamma \vdash g{:}\Phi(\tau\times\sigma_0)\to\tau$.

4. *The triple* $(\sigma_1, M_1, \mathrm{Corec})$ *is a* syntactic corecursive Φ-coalgebra *if*

   (a) $\sigma_1 \in \mathbf{T}$,

   (b) $\vdash f_1{:}\sigma_1\to\Phi(\sigma_1)$,

   (c) $\vdash \mathrm{Corec}{:}\forall\beta.(\beta\to\Phi(\beta+\sigma_1))\to\beta\to\sigma_1$,

   such that
   $$M_1 \circ \mathrm{Corec}\tau g = \Phi([\mathrm{Corec}\tau g, \mathrm{id}]) \circ g$$
   for any $\tau \in \mathbf{T}$ *and* $\Gamma \vdash g{:}\tau\to\Phi(\tau+\sigma_1)$.

We have the following proposition, of which the first part is a syntactic version of a result in [Reynolds and Plotkin 1990] and the second part is a result of [Wraith 1989]. In fact, the first part of the proposition says that the algebraic inductive data types can be represented in $F^{\times+}$, which result originally goes back to [Böhm and Berarducci 1985]. Here we just want to give these representations in short; for further details one may consult [Böhm and Berarducci 1985], [Leivant 1989] or [Girard et al. 1989].

Proposition 3.4 *We work in the system $F^{\times+}$. Let $\Phi(\alpha)$ be a positive type scheme. Then*

1. *There is a syntactic weakly initial Φ-algebra.*

2. *There is a syntactic weakly terminal Φ-coalgebra.*

Proof Let $\Phi(\alpha)$ be a positive type scheme.

1. Define $\sigma_0 := \forall\alpha.(\Phi(\alpha){\to}\alpha){\to}\alpha$, $M_0 := \lambda x{:}\Phi(\sigma).\lambda\alpha.\lambda g{:}\Phi(\alpha){\to}\alpha.g(\Phi(\mathrm{Elim}\alpha g)x)$, and $\mathrm{Elim} := \lambda\alpha.\lambda g{:}\Phi(\alpha){\to}\alpha.\lambda y{:}\sigma.y\alpha g$. Now $(\sigma_0, M_0, \mathrm{Elim})$ is a syntactic weakly initial Φ-algebra.

2. Define $\sigma_1 := \forall\alpha.(\forall\beta.(\beta{\to}\Phi(\beta)){\to}\beta{\to}\alpha){\to}\alpha$, $M_1 := \lambda x{:}\sigma.x(\Phi(\sigma))(\lambda\beta.\lambda g{:}\beta{\to}\Phi(\beta).\lambda z{:}\beta.\Phi(\mathrm{Intro}\beta g)(gx))$, and $\mathrm{Intro} := \lambda\alpha.\lambda g{:}\alpha{\to}\Phi(\alpha).\lambda y{:}\alpha.\lambda\beta.\lambda h{:}\forall(\gamma.\gamma{\to}\Phi(\gamma)){\to}\gamma{\to}\beta.h\alpha g y$. Now $(\sigma_0, M_0, \mathrm{Elim})$ is a syntactic weakly terminal Φ-coalgebra.

It is not known whether there are syntactic recursive algebras or syntactic corecursive coalgebras in $F^{\times+}$. The answer seems to be negative. The well-known definitions of algebraic data-types in $F^{\times+}$ (which are almost the ones defined in the proof above) do in general not allow recursion or corecursion, as will be illustrated by looking at the examples of natural numbers and streams of natural numbers. This means that recursion on Nat and corecursion on Stream have to be defined in terms of iteration on Nat and coiteration on Stream, using the techniques discussed in the Examples 2.7 and 2.8. As was noticed there, it makes a difference whether product and coproduct are weak or semi, so let's note the following fact.

Fact 3.5 *The definable coproduct in $F^{\times+}$ is a weak coproduct, but the definable product in $F^{\times+}$ is a semi product.*
(That is, $<f,g> \circ h = <f\circ h, g\circ h>$, but not $h \circ [f,g] = [h\circ f, h\circ g]$)

Example 3.6 *We define recursive functions on the weak initial algebra of natural numbers.*
Let $(\mathrm{Nat}, M_0, \mathrm{Elim})$ *be the syntactic weak initial algebra of* $\Phi(\alpha) = 1+\alpha$*, as given in the proof of 3.4, where* 1 *and* $+$ *are the second order definable ones. (One can also take the well-known polymorphic Church numerals, which is a slight modification of our type* Nat*. The exposition is not essentially different, but we want to use our categorical understanding of recusion of 2.7.)*
So $\mathrm{Nat} = \forall\alpha((1+\alpha){\to}\alpha){\to}\alpha$, $M_0 = \lambda x{:}1+\mathrm{Nat}.\lambda\alpha.\lambda g.g((\mathrm{id}+\mathrm{Elim}\alpha g)x)$ *and* $\mathrm{Elim} = \lambda\alpha.\lambda g.\lambda y{:}\mathrm{Nat}.y\alpha g$.
Now we first define $\mathbf{Z} := M_0 \circ \mathrm{inl}$ *and* $\S := M_0 \circ \mathrm{inr}$.
Following Example 2.7, we now define $\mathrm{Rec}g = \mathrm{fst}\circ\mathrm{Elim}(\tau\times\mathrm{Nat})(<g, [\mathbf{Z}, \mathbf{S}\circ\mathrm{snd}]>)$*, for* $g{:}1+\tau\times\mathrm{Nat}{\to}\tau$*.*
If

$$g = [g_1, k \circ <\mathrm{fst}, \mathrm{snd}>]$$

for some $k{:}\tau\times\mathrm{Nat}{\to}\tau$*, we obtain the recursion equalities for* $\mathrm{Rec}g$*:*

$$\begin{array}{rcl} \mathrm{Rec}g \circ \mathbf{Z} & & g_1, \\ \mathrm{Rec}g \circ \S^{n+1} \circ \mathbf{Z} & = & \mathrm{inr} \circ g \circ <\mathrm{Rec}g, \mathrm{id}> \circ \S^n \circ \mathbf{Z}. \end{array}$$

(See 2.7 for the restriction on the form of g; the product is semi here.) The predecessor is now defined by taking $g = [\mathbf{Z}, \mathrm{snd}]$*, so* $P := \mathrm{fst}\circ\mathrm{Elim}(\tau\times\mathrm{Nat})(<[\mathbf{Z},\mathrm{snd}], [\mathbf{Z}, \mathbf{S}\circ\mathrm{snd}]>)$*. Notice that* $P(\mathbf{S}t) = t$ *only for standard natural numbers, i.e. for* $t = \S^n(\mathbf{Z}*)$*, with* $*$ *the unique (closed) term of type* 1*. Also notice that* P *computes the predecessor of a natural number* n *in a number of steps of order* n*.*

Example 3.7 *We define corecursive functions on streams of natural numbers. Take for* Stream *the syntactic weakly terminal* Φ*-coalgebra as in the proof of 3.4, for* $\Phi(\alpha) = \mathrm{Nat}\times\alpha$*. So* $\mathrm{Stream} = \forall\alpha.(\forall\beta.(\beta{\to}(\mathrm{Nat}{\times}\beta)){\to}\beta{\to}\alpha){\to}\alpha$, $M_1 = \lambda x{:}\mathrm{Stream}.x(\mathrm{Nat}{\times}\sigma)(\lambda\beta.\lambda g{:}\beta{\to}\mathrm{Nat}{\times}\beta.\lambda z{:}\beta.(\mathrm{id}{\times}\mathrm{Intro}\beta g)(gx))$*, and* $\mathrm{Intro} = \lambda\alpha.\lambda g{:}\alpha{\to}\mathrm{Nat}\times\alpha.\lambda y{:}\alpha.\lambda\beta.\lambda h{:}\forall(\gamma.\gamma{\to}\mathrm{Nat}\times\gamma){\to}\gamma{\to}\beta.h\alpha g y$*. We can define head and tail functions by taking* $\mathbf{H} := \mathrm{fst}\circ M_1$ *and* $\mathbf{T} := \mathrm{snd}\circ M_1$*. Following Example 2.8, we now define for* $g{:}\tau{\to}\mathrm{Nat}\times(\tau+\mathrm{Stream})$ $\mathrm{Corec}g := \mathrm{Intro}(\tau+\mathrm{Stream})([g, <H, \mathrm{inr}\circ\mathbf{T}>])\circ\mathrm{inl}$*. As the coproduct is not semi, but weak (see 2.8), we find that only for* $g = <g_1, \mathrm{in}\circ k>$ *for* $\in$ *is* inr *or* inl *and some* $k{:}\mathrm{Stream}{\to}B$ *or* $k{:}\mathrm{Stream}{\to}\mathrm{Stream}$ *we obtain the corecursion equations.*

$$\begin{array}{rcl} \mathbf{H}\circ\mathrm{Corec}g & = & g_1, \\ \mathbf{T}\circ\mathrm{Corec}g & = & [\mathrm{Corec}g, \mathrm{id}]\circ\mathrm{snd}\circ g. \end{array}$$

The function that replaces the head of a stream by zero is now defined by $\mathrm{ZeroH} := \mathrm{Corec}<\mathbf{Z}, \mathrm{inr}\circ\mathbf{T}>$

It is really impossible to define a 'global' predecessor on the weakly initial natural numbers as described above (and similarly for the polymorphic Church numerals.) Also it is impossible to define a global ZeroH-function on the weakly terminal streams as described above. This is shown in the following proposition.

Proposition 3.8 *1. For* $\mathrm{Nat} = \forall\alpha((1+\alpha)\to\alpha)\to\alpha$*, there is no closed term* $P{:}\mathrm{Nat}\to\mathrm{Nat}$ *such that* $P(\mathbf{S}x) = x$ *for x a variable.*

2. *For* $\mathrm{Stream} := \forall\beta.(\forall\gamma.(\gamma\to\mathrm{Nat})\to(\gamma\to\gamma)\to\gamma\to\beta)\to\beta$ *there is no closed term* $\mathrm{ZeroH}{:}\mathrm{Stream}\to\mathrm{Stream}$ *such that* $\mathbf{T}(\mathrm{ZeroH}y) = \mathbf{T}y$ *and* $\mathbf{H}(\mathrm{ZeroH}y) = 0$ *for y a variable.*

Proof Both cases immediately by the Churh-Rosser property for the system $F^{\times+}$.

One can show in general that the (co)inductive types in system $F^{\times+}$ as defined above do not allow (co)recursion, i.e. they are weakly initial (terminal) (co)algebras.

4 Recursive algebras and corecursive coalgebras in models of polymorphic lambda calculus

We now want to study the interrelations between polymorphism, recursive T-algebras and corecursive T-coalgebras for syntactically definable functors T. We therefore look at a slight extension of the so called K-models defined by [Reynolds and Plotkin 1990] in which there is a notion of *expressible functor*. The original notion of K-model is quite weak and just strong enough to show that there are no **Set**-models (See [Reynolds and Plotkin 1990], or [Reynolds 1984], where the result was first discussed.) The strengthening we propose is very natural, just saying that the type-β-reduction is sound and that we have some substitution properties. To be able to describe corecursive coalgebras we have to require that K is not just a ccc, but that it also has coproducts. So let in the following K be a ccc with coproducts. We first introduce some definitions that will be used later in the model definition.

Definition 4.1 *Let A, B, C be objects of K.*

1. *Define* $\phi_{A,B,C} : (A \to C^B) \to (A \times B \to C)$ *by*

$$\phi_{A,B,C}(g) = \mathsf{ev} \circ (g \times \mathsf{id})$$

and $\phi^{-}_{A,B,C} : (A \times B \to C) \to (A \to C^B)$ *by*

$$\phi^{-}_{A,B,C}(f) = \Lambda(f)$$

This gives an isomorphism between the homsets $A \to C^B$ *and* $A \times B \to C$*;* $\phi \circ \phi^{-} = Id$ *and* $\phi^{-} \circ \phi = Id$*.*

2. *From ϕ we define* $\psi_{A,B,C} : (A \to C^B) \simeq (B \to C^A)$ *by* $\psi_{A,B,C}(g) = \phi^{-}_{B,A,C}(\phi_{A,B,C}(g) \circ \langle\pi_2, \pi_1\rangle) = \Lambda(\phi_{A,B,C}(g) \circ \langle\pi_2, \pi_1\rangle)$*. This gives an isomorphism by* $\psi_{A,B,C} \circ \psi_{B,A,C} = Id$*.*

3. *If $A = 1$ we also define the isomorphism* $k_{B,C} : (1 \to C^B) \simeq (B \to C)$ *by* $k_{B,C}(g) = \phi(g) \circ \langle !, \mathsf{id}\rangle$ *and* $k^{-}_{B,C}(f) = \phi^{-}(f \circ \pi_2)$*.*

Definition 4.2 ([Reynolds and Plotkin 1990]) *Given a category K, a* type assignment *in K is a mapping from the set TypVar to the objects of the category K. A K-*model *consists of a triple* $(K, \{\!|-|\!\}, [\![-]\!])$*, where*

1. *K is a cartesian closed category with coproducts.*

2. *For each type assignment ξ there is a mapping* $\{\!|-|\!\}_\xi$ *from* $\mathbf{T}$ *to the objects of K such that*

   (a) $\{\!|\alpha|\!\}_\xi = \xi(\alpha)$*,*

(b) $\{\!|\sigma\to\tau|\!\}_\xi = \{\!|\tau|\!\}_\xi^{\{\!|\sigma|\!\}_\xi}$,

(c) $\{\!|\sigma\times\tau|\!\}_\xi = \{\!|\sigma|\!\}_\xi \times \{\!|\tau|\!\}_\xi$,

(d) $\{\!|\sigma+\tau|\!\}_\xi = \{\!|\sigma|\!\}_\xi + \{\!|\tau|\!\}_\xi$,

(e) if $\xi\lceil FTV(\sigma) = \xi'\lceil FTV(\sigma)$, *then* $\{\!|\sigma|\!\}_\xi = \{\!|\sigma|\!\}_{\xi'}$.

3. *For each type assignment* ξ, *type* σ *and context* $\Gamma = x_1{:}\sigma_1, \ldots x_n{:}\sigma_n$ *there is a mapping* $[\![-]\!]_\xi^{\sigma\Gamma}$ *from* $Term(\sigma,\Gamma)$ *to* $(1\times\{\!|\sigma_1|\!\}_\xi \times\ldots\times\{\!|\sigma_n|\!\}_\xi) \to \{\!|\sigma|\!\}_\xi$. *such that*

   (a) $[\![x_i]\!]_\xi^{\sigma\Gamma} = \pi_2\circ\pi_1^{n-i}$,

   (b) $[\![\mathrm{fst}^{\sigma_1\sigma_2}]\!]_\xi^{\sigma_1\times\sigma_2\to\sigma_1\Gamma} = \Lambda(\pi_1\circ\pi_2)$ *and* $[\![\mathrm{snd}^{\sigma_1\sigma_2}]\!]_\xi^{\sigma_1\times\sigma_2\to\sigma_2\Gamma} = \Lambda(\pi_2\circ\pi_2)$,

   (c) $[\![\mathrm{inl}^{\sigma_1\sigma_2}]\!]_\xi^{\sigma_1\to\sigma_1+\sigma_2\Gamma} = \Lambda(\mathsf{in}_1\circ\pi_2)$ *and* $[\![\mathrm{inr}^{\sigma_1\sigma_2}]\!]_\xi^{\sigma_2\to\sigma_1+\sigma_2\Gamma} = \Lambda(\mathsf{in}_2\circ\pi_2)$,

   (d) if $\Gamma\vdash M{:}\sigma\to\tau$ *and* $\Gamma\vdash N{:}\sigma$,*then* $[\![MN]\!]_\xi^{\tau\Gamma} = \mathsf{ev}\circ\langle[\![M]\!]_\xi^{\sigma\to\tau\Gamma}, [\![N]\!]_\xi^{\sigma\Gamma}\rangle$,

   (e) if $\Gamma, x{:}\sigma\vdash N{:}\tau$, *then* $[\![\lambda x{:}\sigma.M]\!]_\xi^{\sigma\to\tau\Gamma} = \Lambda([\![M]\!]_\xi^{\tau\Gamma,x:\sigma})$,

   (f) if $\Gamma\vdash M{:}\rho\to\sigma$ *and* $\Gamma\vdash N{:}\rho\to\tau$, *then* $[\![<M,N>]\!]_\xi^{\rho\to\sigma\times\tau\Gamma} = \phi^-(\langle\phi([\![M]\!]_\xi^{\rho\to\sigma\Gamma}), \phi'([\![N]\!]_\xi^{\rho\to\tau\Gamma})\rangle)$, *where* ϕ, ϕ' *and* ϕ^- *are the appropriate isomorphisms of 4.1.*

   (g) if $\Gamma\vdash M{:}\sigma\to\rho$ *and* $\Gamma\vdash N{:}\tau\to\rho$, *then* $[\![[M,N]]\!]_\xi^{\sigma+\tau\to\rho\Gamma} = \psi([\psi'([\![M]\!]_\xi^{\rho\to\sigma\Gamma}), \psi''([\![N]\!]_\xi^{\rho\to\tau\Gamma})])$, *where* ψ, ψ' *and* ψ'' *are the appropriate isomorphisms of 4.1.*

   (h) if $\xi\lceil FTV(\sigma,\Gamma) = \xi'\lceil FTV(\sigma,\Gamma)$, *then* $[\![M]\!]_\xi^{\sigma\Gamma} = [\![M]\!]_{\xi'}^{\sigma\Gamma}$,

   (i) if $\Gamma(= x_1{:}\sigma_1,\ldots,x_n{:}\sigma_n) \subset \Gamma'(= y_1{:}\tau_1,\ldots,y_m{:}\tau_m)$ *this gives rise to a canonical morphism* $\langle\pi_1,\pi_{j_1},\ldots,\pi_{j_n}\rangle : 1\times\tau_1\times\ldots\times\tau_m \to 1\times\sigma_1\times\ldots\times\sigma_n$; *now if* $\Gamma\vdash M{:}\sigma$ *then* $[\![M]\!]_\xi^{\sigma\Gamma'} = [\![M]\!]_\xi^{\sigma\Gamma}\circ\langle\pi_1,\pi_{j_1},\ldots,\pi_{j_n}\rangle$,

   (j) if $\Gamma\vdash(\lambda\alpha.M)\sigma{:}\tau$, *then* $[\![(\lambda\alpha.M)\sigma]\!]_\xi^{\tau\Gamma} = [\![M[\sigma/\alpha]]\!]_\xi^{\tau\Gamma}$.

   (k) if $\Gamma, x{:}\sigma\vdash M{:}\tau$ *and* $\Gamma\vdash N{:}\sigma$, *then* $[\![M[N/x]]\!]_\xi^{\tau\Gamma} = [\![M]\!]_\xi^{\tau\Gamma,x:\sigma}\circ\langle\mathsf{id}, [\![N]\!]_\xi^{\tau\Gamma}\rangle$,

   (l) if $\Gamma\vdash M{:}\tau$,*then*$[\![M[\sigma/\alpha]]\!]_\xi^{\tau[\sigma/\alpha]\Gamma[\sigma/\alpha]} = [\![M]\!]_{\xi(\alpha:=\{\!|\sigma|\!\}_\xi)}^{\tau\Gamma}$.

If there is no ambiguity, all sub- and superscripts will be omitted.

To compute with K-models we give, without proof, some properties of the interpretation function.

Lemma 4.3 *1. For* $\vdash h{:}\sigma\to\mu$, $\vdash g{:}\tau\to\mu$ *and* $\vdash l{:}\mu\to\rho$, $k([\![l\circ[h,g]]\!]_\xi) = k([\![[l\circ h, l\circ g]]\!]_\xi)$.

2. *For* $\vdash h{:}\mu\to\sigma$, $\vdash g{:}\mu\to\tau$ *and* $\vdash l{:}\rho\to\mu$, $k([\![<h,g>\circ l]\!]_\xi) = k([\![<h\circ l, g\circ l>]\!]_\xi)$.

3. $k([\![\mathrm{id}_\sigma + \mathrm{id}_\tau]\!]_\xi) = k([\![\mathrm{id}_{\sigma+\tau}]\!]_\xi) = \mathsf{id}_{\{\!|\sigma+\tau|\!\}_\xi}$.

4. $k([\![\mathrm{id}_\sigma \times \mathrm{id}_\tau]\!]_\xi) = k([\![\mathrm{id}_{\sigma\times\tau}]\!]_\xi) = \mathsf{id}_{\{\!|\sigma\times\tau|\!\}_\xi}$

This lemma states that, although the syntactic product and coproduct are weak, they behave as semi-product and semi-coproduct under the interpretation. The Lemma implies that $[\![\ldots[h,g]\circ l\ldots]\!]_\xi = [\![\ldots[h\circ l, g\circ l]\ldots]\!]_\xi$, $[\![\ldots\mathrm{id}_\sigma+\mathrm{id}_\tau\ldots]\!]_\xi = [\![\ldots\mathrm{id}_{\sigma+\tau}\ldots]\!]_\xi$ and so forth.

5 Expressibility of functors and (co)recursive (co)algebras in K-models

Definition 5.1 ([Reynolds and Plotkin 1990]) *Given a* K*-model* $(K, \{\!|-|\!\}, [\![-]\!])$, *a (covariant) functor* $F: K\to K$ *is* expressible *if there is a positive type scheme* $\Phi(\alpha)$ *such that*

1. *For each* $\sigma \in \mathbf{T}$
$$\{\!\!\{\Phi(\sigma)\}\!\!\}_\xi = F(\{\!\!\{\sigma\}\!\!\}_\xi).$$

2. *For each* $\sigma, \tau \in \mathbf{T}$ *and* $\Gamma \vdash M{:}\sigma{\to}\tau$
$$k([\![\Phi(M)]\!]_\xi^{\Phi(\sigma)\to\Phi(\tau)\Gamma} \circ f) = F(k'([\![M]\!]_\xi^{\sigma\to\tau\Gamma} \circ f)),$$
for any $f : 1 \to 1 \times \{\!\!\{\sigma_1\}\!\!\}_\xi \times \ldots \times \{\!\!\{\sigma_n\}\!\!\}_\xi$ *(if* $\Gamma = x_1{:}\sigma_1, \ldots x_n{:}\sigma_n$*) where* k *and* k' *are the appropriate isomorphisms of 4.1.*

For expressible functors in a K-model we also have a notion of expressible weakly initial algebra (terminal coalgebra) and expressible recursive algebra (corecursive coalgebra.)

Definition 5.2 *Let* M *be a* K*-model and* F *a functor expressible by* Φ *in* M.

1. *The triple* (A, f, Elim) *is an expressible weakly initial* F*-algebra if there is a closed type* σ*, a closed term* $M{:}\Phi(\sigma){\to}\sigma$ *and a closed term* $h{:}\forall\alpha.(\Phi(\alpha){\to}\alpha){\to}\sigma{\to}\alpha$ *such that*
   (a) $\{\!\!\{\sigma\}\!\!\}_\xi = A$,
   (b) $k([\![M]\!]) = f$,
   (c) *for all* $g{:}F(B){\to}B$, $\mathsf{Elim} g = k([\![h\alpha y]\!]_{\xi(\alpha:=B)} \circ \langle \mathsf{id}, k^-(g)\rangle)$.

2. *The pair* (A, f, Intro) *is an expressible weakly terminal* F*-coalgebra if there is a closed type* σ*, a closed term* $M{:}\sigma{\to}\Phi(\sigma)$ *and a closed term* $h{:}\forall\alpha.(\alpha{\to}\Phi(\alpha)){\to}\alpha{\to}\sigma$ *such that*
   (a) $\{\!\!\{\sigma\}\!\!\}_\xi = A$,
   (b) $k([\![M]\!]) = f$,
   (c) *for all* $g{:}B{\to}F(B)$, $\mathsf{Intro} g = k([\![h\alpha y]\!]_{\xi(\alpha:=B)} \circ \langle \mathsf{id}, k^-(g)\rangle)$.

3. *The pair* (A, f, Rec) *is an expressible recursive* F*-algebra if there is a closed type* σ*, a closed term* $M{:}\Phi(\sigma){\to}\sigma$ *and a closed term* $h{:}\forall\alpha.(\Phi(\alpha \times \sigma){\to}\alpha){\to}\sigma{\to}\alpha$ *such that*
   (a) $\{\!\!\{\sigma\}\!\!\}_\xi = A$,
   (b) $k([\![M]\!]) = f$,
   (c) *for all* $g{:}F(B \times A){\to}B$, $\mathsf{Rec} g = k([\![h\alpha y]\!]_{\xi(\alpha:=B)} \circ \langle \mathsf{id}, k^-(g)\rangle)$.

4. *The pair* (A, f, Corec) *is an expressible corecursive* F*-coalgebra if there is a closed type* σ*, a closed term* $M{:}\sigma{\to}\Phi(\sigma)$ *and a closed term* $h{:}\forall\alpha.(\alpha{\to}\Phi(\alpha + \sigma)){\to}\alpha{\to}\sigma$ *such that*
   (a) $\{\!\!\{\sigma\}\!\!\}_\xi = A$,
   (b) $k([\![M]\!]) = f$,
   (c) *for all* $g{:}B{\to}F(B + A)$, $\mathsf{Corec} g = k([\![h\alpha y]\!]_{\xi(\alpha:=B)} \circ \langle \mathsf{id}, k^-(g)\rangle)$.

In all these cases we say that the weakly initial (terminal) (co)algebra or (co)recursive (co)algebra is expressed by (σ, M, h).

Lemma 5.3 *Let the functor* F *be expressed by* Φ *in the* K*-model* M*. Then*

1. *If* (σ_0, M_0, h_0) *expresses a weakly initial* F*-algebra, then*
$$[\![h_0\tau y \circ M_0]\!]^{y:\Phi(\tau)\to\tau} = [\![y \circ \Phi(h_0\tau y)]\!]^{y:\Phi(\tau)\to\tau}.$$

2. *If* (σ_1, M_1, h_1) *expresses a weakly terminal* F*-coalgebra, then*
$$[\![M_1 \circ h_1\tau y]\!]^{y:\tau\to\Phi(\tau)} = [\![\Phi(h_1\tau y) \circ y]\!]^{y:\tau\to\Phi(\tau)}.$$

3. *If (σ_0, M_0, h_0) expresses a recursive F-algebra, then*
$$[\![h_0\tau y \circ M_0]\!]^{y:\Phi(\tau\times\sigma)\to\tau} = [\![y \circ \Phi(<h_0\tau y, \mathrm{id}>)]\!]^{y:\Phi(\tau\times\sigma)\to\tau}.$$

4. *If (σ_1, M_1, h_1) expresses a corecursive F-coalgebra, then*
$$[\![M_1 \circ h_1\tau y]\!]^{y:\tau\to\Phi(\tau+\sigma)} = [\![\Phi([h_1\tau y, \mathrm{id}]) \circ y]\!]^{y:\tau\to\Phi(\tau+\sigma)}.$$

Proof Just by writing out the definitions, using the properties of the interpretation, like the ones in Lemma 4.3

The following proposition states the existence of expressible weakly initial F-algebras and expressible weakly terminal F-coalgebras in a K-model for an expressible functor F. The first part was shown by [Reynolds and Plotkin 1990] and the second part follows from a similar argument, using the work of [Wraith 1989]. The proposition follows immediately from Proposition 3.4: If F is expressed by the type scheme Φ and Φ has a syntactic weakly initial algebra (σ, M, h), then the the interpretation of (σ, M, h) in M is a weakly initial algebra expressible by (σ, M, h).

Proposition 5.4 *Let M be a K-model and F an expressible functor in M.*

1. M *has an expressible weakly initial F-algebra,*
2. M *has an expressible weakly terminal F-coalgebra.*

In general, K-models need not have all recursive algebras and all corecursive coalgebras, but we do have the following, which is our central Theorem, linking recursive algebras with corecursive coalgebras via polymorphism.

Theorem 5.5 *Let M be a K-model.Then the following two statements are equivalent.*

1. *All expressible functors F have an expressible corecursive F-coalgebra.*
2. *All expressible functors F have an expressible recursive F-algebra.*

Proof The proof falls in two parts. First assume that all expressible functors have an expressible corecursive coalgebra. We show that if F is expressed by Φ, then also
$$T(X) = \{\!|\forall\beta.(\Phi(\beta\times\alpha)\to\beta)\to\beta)|\!\}_{(\alpha:=X)}$$
is expressible. From the expressible corecursive T-coalgebra (A, f_1, Corec) we define an expressible recursive F-algebra, with the same object A as carrier.
Then assume that all expressible functors have an expressible recursive algebra. Now if F is an expressible functor, then also
$$T(X) = \{\!|\exists\beta.(\beta\to\Phi(\beta+\alpha))\times\beta|\!\}_{(\alpha:=X)}$$
is expressible. From the expressible recursive T-algebra (A, f_0, Rec) we define an expressible corecursive F-coalgebra, with the same object A as carrier. (Note that $\exists\beta.(\beta\to\Phi(\beta+\alpha))\times\beta$ is defined by $\forall\gamma.(\forall\beta.(\beta\to\Phi(\beta+\alpha)\times\beta)\to\gamma)\to\gamma$.)
Only the first part will be done in some more detail; for the second part we only show how to define the expressible corecursive F-coalgebra from the recursive T-algebra. To check that what we define really satisfies the requirements is quite similar to the first case.
Let's now suppose that all expressible functors have an expressible corecursive coalgebra and let F be a functor, expressible by Φ. Now
$$T(X) = \{\!|\forall\beta.(\Phi(\beta\times\alpha)\to\beta)\to\beta)|\!\}_{(\alpha:=X)}$$
is expressible by Θ, given by
$$\Theta(\alpha) = \forall\beta.(\Phi(\beta\times\alpha)\to\beta)\to\beta)$$
For $f{:}\sigma\to\tau$ we have
$$\Theta(f) = \lambda x{:}\Theta(\sigma).\lambda\beta.\lambda h{:}\Phi(\beta\times\tau)\to\beta.x\beta(\lambda y{:}\Phi(\beta\times\sigma).h(\Phi(\mathrm{id}\times f)y)).$$
Now T has an expressible corecursive coalgebra, say (A, f_1, Corec), expressed by (σ, M_1, h_1). That is

1. $A = \{\!|\sigma|\!\}$
2. $f_1 = k([\![M_1]\!])$
3. $\mathsf{Corec} g = k([\![h_1 \alpha y]\!]_{(\alpha:=X)} \circ \langle \mathsf{id}, k^-(g)\rangle)$ for any $g{:}X \to T(X + A)$.

We define

$$\begin{aligned} h_0 &:= \lambda\alpha.\lambda g{:}\Phi(\alpha\times\sigma)\to\alpha.\lambda x{:}\sigma.M_1 x\alpha g,\\ M_0 &:= h_1(\Phi\sigma)(\lambda z{:}\Phi(\sigma).\lambda\beta.\lambda h{:}\Phi(\beta\times\Phi(\sigma))\to\beta.h(\Phi(<h_0\beta(h\circ\Phi(\mathrm{id}\times\mathrm{inr})),\mathrm{inr}>)z)). \end{aligned}$$

Now we claim that (σ, M_0, h_0) expresses a recursive F-algebra, namely (A, f_0, Rec), where $f_0 := k([\![M_0]\!])$ and $\mathsf{Rec} g = k([\![h_0\alpha y]\!]_{(\alpha:=X)} \circ \langle \mathsf{id}, k^-(g)\rangle)$ for any $g{:}F(X\times A) \to X$. By definition (A, f_0, Rec) is expressed by (σ, M_0, h_0). That (A, f_0, Rec) is a recursive F-algebra is shown as follows. Let $g{:}F(X\times A)\to X$.

$$\begin{aligned} \mathsf{Rec} g\circ f_0 &= k([\![h_0\alpha y]\!]^{y:\Phi(\alpha\times\sigma)\to\alpha}_{(\alpha:=X)} \circ \langle \mathsf{id}, k^-(g)\rangle)\circ k([\![M_0]\!])\\ &= k([\![h_0\alpha y\circ M_0]\!]^{y:\Phi(\alpha\times\sigma)\to\alpha}_{(\alpha:=X)} \circ \langle \mathsf{id}, k^-(g)\rangle)\\ &\quad \text{writing } g_1 \text{ for } (\lambda z{:}\Phi(\sigma).\lambda\beta.\lambda h{:}\Phi(\beta\times\Phi(\sigma))\to\beta.h(\Phi(<h_0\beta(h\circ\Phi(\mathrm{id}\times\mathrm{inr})),\mathrm{inr}>)z))\\ &= k([\![\lambda x{:}\Phi(\sigma).M_1(h_1(\Phi\sigma)g_1x)\alpha y]\!]^{y:\Phi(\alpha\times\sigma)\to\alpha}_{(\alpha:=X)} \circ \langle \mathsf{id}, k^-(g)\rangle)\\ &= k([\![\lambda x{:}\Phi(\sigma).\Theta(\big[h_1(\Phi\sigma)g_1,\mathrm{id}\big])(g_1x)\alpha y]\!]^{y:\Phi(\alpha\times\sigma)\to\alpha}_{(\alpha:=X)} \circ \langle \mathsf{id}, k^-(g)\rangle)\\ &= k([\![y\circ\Phi(<h_0\alpha y,\mathrm{id}>)]\!]^{y:\Phi(\alpha\times\sigma)\to\alpha}_{(\alpha:=X)} \circ \langle \mathsf{id}, k^-(g)\rangle)\\ &= g\circ F(\langle \mathsf{Rec}(g), \mathsf{id}\rangle) \end{aligned}$$

Here the Lemmas 4.3 and 5.3 are of course heavily used.
Let's now suppose that all expressible functors have an expressible recursive algebra and let F be a functor, expressible by Φ. Now

$$T(X) = \{\!|\forall\gamma.(\forall\beta.(\beta\to\Phi(\beta+\alpha)\times\beta)\to\gamma)\to\gamma|\!\}_{(\alpha:=X)}$$

is expressible by Θ, given by

$$\Theta(\alpha) = \forall\gamma.(\forall\beta.(\beta\to\Phi(\beta+\alpha)\times\beta)\to\gamma)\to\gamma.$$

For $f{:}\sigma\to\tau$ we have

$$\Theta(f) = \lambda x{:}\Theta(\sigma).\lambda\gamma.\lambda h{:}\forall\beta.(\beta\to\Phi(\beta+\tau)\times\beta)\to\gamma.x\gamma(\lambda\beta.\lambda y{:}\beta\to\Phi(\beta+\sigma)\times\beta.h\beta<\Phi(\mathrm{id}+f)\circ y_1, y_2>),$$

where y_1 denotes $\mathrm{fst}y$ and y_2 denotes $\mathrm{snd}y$. There is an expressible recursive T-algebra, say (A, f_0, Rec), expressed by (σ, M_0, h_0), so we have

1. $A = \{\!|\sigma|\!\}$
2. $f_0 = k([\![M_0]\!])$
3. $\mathsf{Rec} g = k([\![h_0\alpha y]\!]_{(\alpha:=X)} \circ \langle \mathsf{id}, k^-(g)\rangle)$ for any $g{:}T(X\times A)\to X$.

Now define

$$\begin{aligned} h_1 &:= \lambda\alpha.\lambda g{:}\alpha\to\Phi(\alpha+\sigma).\lambda x{:}\alpha.M_0(\lambda\gamma.\lambda h{:}\forall\beta.(\beta\to\Phi(\beta+\sigma)\times\beta)\to\gamma.h\alpha<g,x>),\\ M_1 &:= h_0(\Phi(\sigma))(\lambda z{:}\Theta(\Phi(\sigma)\times\sigma).z(\Phi(\sigma)(\lambda\gamma.\lambda p.\Phi([h_1\beta(\Phi(\mathrm{id}+\mathrm{snd})\circ p_1,\mathrm{snd}])(p_1p_2))), \end{aligned}$$

where $p{:}(\beta\to\Phi(\beta+(\Phi(\sigma)\times\sigma)))\times\beta$, p_1 and p_2 abbreviate $\mathrm{fst}p$ and $\mathrm{snd}p$. The triple (σ, M_1h_1) expresses a corecursive F-coalgebra.

6 Syntax for recursive algebras and corecursive coalgebras

In fact, what Theorem 5.5 boils down to is the syntactic interdefinability of recursive algebras and corecursive coalgebras in a polymorphic typed lambda calculus. To make that precise we shall define an extension of $F^{\times +}$, which includes a syntactical formalization of recursive algebras and corecursive coalgebras, and show that one can define recursive algebras in terms of corecursive coalgebras and vice versa. We have not studied the proof theory of this syntax but we do want to describe the system, as it makes clear what is going on in the proof of 5.5.
We then want to relate Theorem 5.5 to a system of recursive (and corecursive) types described by [Mendler 1987]. The syntax there is a bit too weak to define the recursive types in terms of the corecursive types (or the other way around), but we show that in K-models of this syntax we do have recursive algebras and corecursive coalgebras for all expressible functors. This is done by showing that the system has syntactic recursive Φ-algebras and syntactic corecursive Φ-coalgebras for every positive type scheme Φ. (See Definition 3.3.)

Definition 6.1 *The system $F^{\times +}_{(co)rec}$ is the system $F^{\times +}$ extended with the following.*

1. *The set of types $\mathbf{T}$ is extended with $\mu\alpha.\Phi(\alpha)$ and $\nu\alpha.\Phi(\alpha)$, for $\Phi(\alpha)$ a positive type scheme.*
2. *For $\mu\alpha.\Phi(\alpha)$ and $\nu\alpha.\Phi(\alpha)$ we have the extra constants*

$$\begin{array}{rl} \mathbf{In}_\mu : \Phi(\mu)\to\mu & \mathrm{Rec}_\mu : \forall\alpha.(\Phi(\alpha\times\mu)\to\alpha)\to\mu\to\alpha, \\ \mathbf{Out}_\nu : \nu\to\Phi(\nu) & \mathrm{Corec}_\nu : \forall\alpha.(\alpha\to\Phi(\alpha+\nu))\to\alpha\to\nu. \end{array}$$

3. *Reduction rules for μ and ν:*

$$\begin{array}{rcl} \mathrm{Rec}\tau g(\mathbf{In}x) & \longrightarrow_\mu & g(\Phi(<\mathrm{Rec}\tau g, \mathrm{id}>)x), \\ \mathbf{Out}(\mathrm{Corec}\tau gx) & \longrightarrow_\nu & \Phi([\mathrm{Corec}\tau g, \mathrm{id}])(gx). \end{array}$$

4. *Extra reduction rules for $\times$ and $+$:*

$$\begin{array}{rcl} [\mathrm{inl}, \mathrm{inr}] & \longrightarrow_+ & \mathrm{id}, \\ <\mathrm{fst}, \mathrm{snd}> & \longrightarrow_\times & \mathrm{id}, \\ f\circ[g,h] & \longrightarrow_+ & [f\circ g, f\circ h], \\ <g,h>\circ f & \longrightarrow_\times & <g\circ f, h\circ f>. \end{array}$$

(μ abbreviates $\mu\alpha.\Phi(\alpha)$ and ν abbreviates $\nu\alpha.\Phi(\alpha)$.)

Note that with these extra reduction rules for product and coproduct, all positive type schemes become covariant functors and all negative type schemes become contravariant functors. (See the remark before 3.3.)
We can give the following syntactical formulation of Theorem 5.5 in this new system $F^{\times +}_{(co)rec}$.

Proposition 6.2 *In $F^{\times +}_{(co)rec}$ we can define ν,* **Out** *and* Corec *in terms of μ,* **In** *and* Rec *and vice versa.*

Proof Suppose we only have the rules for μ, **In** and Rec and let Φ be a positive type scheme. Define

$$\begin{array}{rcl} \Theta(\alpha) & := & \forall\gamma.(\forall\beta.(\beta\to\Phi(\beta+\alpha)\times\beta)\to\gamma)\to\gamma, \\ \sigma & := & \mu\alpha.\Theta(\alpha), \\ \mathrm{Corec} & := & \lambda\alpha.\lambda g{:}\alpha\to\Phi(\alpha+\sigma).\lambda x{:}\alpha.\mathbf{In}_\sigma(\lambda\gamma.\lambda h{:}\forall\beta.(\beta\to\Phi(\beta+\sigma)\times\beta)\to\gamma.h\alpha<g,x>), \\ \mathbf{Out} & := & \mathrm{Rec}_\sigma(\Phi(\sigma))(\lambda z{:}\Theta(\Phi(\sigma)\times\sigma).z(\Phi(\sigma))(\lambda\gamma.\lambda p.\Phi([h_1\beta(\Phi(\mathrm{id}+\mathrm{snd})\circ p_1, \mathrm{snd}])(p_1p_2))), \end{array}$$

where p_1 and p_2 abbreviate fst$p$ and snd$p$. Then

$$\mathbf{Out}(\mathrm{Corec}\tau g x) \twoheadrightarrow \Phi([\mathrm{Corec}\tau g, \mathrm{id}])(gx)).$$

The other way around, suppose we only have rules for ν, **Out** and Corec and let Φ be a positive type scheme. Define

$$\begin{array}{rcl}\Theta(\alpha) & = & \forall\beta.(\Phi(\beta\times\alpha)\to\beta)\to\beta,\\ \sigma & := & \nu\alpha.\Theta(\alpha),\\ \mathrm{Rec} & := & \lambda\alpha\lambda g{:}\Phi(\alpha\times\sigma)\to\alpha.\lambda x{:}\sigma.\mathbf{Out}_\sigma x\alpha g,\\ \mathbf{In} & := & \mathrm{Corec}_\sigma(\Phi(\sigma))(\lambda z{:}\Phi(\sigma).\lambda\beta.\lambda h{:}\Phi(\beta\times\Phi(\sigma))\to\beta.h(\Phi(<h_0\beta(h\circ\Phi(\mathrm{id}\times\mathrm{inr})),\mathrm{inr}>)z)).\end{array}$$

Then

$$\mathrm{Rec}\tau g(\mathbf{In}x) \twoheadrightarrow g(\Phi(<\mathrm{Rec}\tau g, \mathrm{id}>)x).$$

We now want to look at the system of recursive types, as defined by [Mendler 1987], let's call it $F_{(CO)REC}$. (The system also has corecursive types.)

Definition 6.3 ([Mendler 1987]) *The system $F_{(CO)REC}$ is defined by adding to the polymorphic lambda calculus the following.*

1. *The set of types* $\mathbf{T}$ *is extended with $\mu\alpha.\Phi(\alpha)$ and $\nu\alpha.\Phi(\alpha)$, for $\Phi(\alpha)$ a positive type scheme.*
2. *For $\mu\alpha.\Phi(\alpha)$ and $\nu\alpha.\Phi(\alpha)$ we have the extra constants*

$$\begin{array}{ll}\mathbf{In}_\mu : \Phi(\mu)\to\mu & \mathrm{R}_\mu : \forall\beta.(\forall\gamma.(\gamma\to\mu)\to(\gamma\to\beta)\to\Phi(\gamma)\to\beta)\to\mu\to\beta,\\ \mathbf{Out}_\nu : \nu\to\Phi(\nu) & \Omega_\nu : \forall\beta.(\forall\gamma.(\nu\to\gamma)\to(\beta\to\gamma)\to\beta\to\Phi(\gamma))\to\beta\to\nu.\end{array}$$

3. *Reduction rules for μ and ν:*

$$\begin{array}{rcl}\mathrm{R}_\mu\tau g(\mathbf{In}_\mu x) & \longrightarrow_\mu & g\mu(\mathrm{id}_\mu)(\mathrm{R}_\mu\tau g)x),\\ \mathbf{Out}_\nu(\Omega_\nu\tau g x) & \longrightarrow_\nu & g\nu(\mathrm{id}_\nu)(\Omega_\nu\tau g)x.\end{array}$$

(μ abbreviates $\mu\alpha.\Phi(\alpha)$ and ν abbreviates $\nu\alpha.\Phi(\alpha)$.)

In [Mendler 1987] it is shown that this system satisfies a lot of nice properties, like strong normalization and confluence of the reduction relation.

Definition 6.4 *The system $F_{(CO)REC}$ with only the rules for μ will be called F_{REC} and similarly, $F_{(CO)REC}$ with only the rules for ν will be called F_{COREC}.*

We now want to show that in K-models of F_{REC}, all expressible functors have a recursive algebra (and hence a corecursive coalgebra) and that in K-models of F_{COREC}, all expressible functors have a corecursive coalgebra (and hence a recursive algebra.) First the (straightforward) definition of K-model for F_{REC}, respectively F_{COREC}.

Definition 6.5 1. *A K-model for polymorphic lambda calculus M is a model for F_{REC} if for every positive type scheme Φ and object valuation ξ, there are*

   (a) an object A_ξ,

   (b) a morphism $\mathsf{In}_\xi : 1 \to A_\xi^{(\Phi(\alpha))_{\xi(\alpha:=A_\xi)}}$,

   (c) a morphism $\mathsf{R}_\xi : 1 \to (\forall\beta.(\forall\gamma.(\gamma\to\alpha)\to(\gamma\to\beta)\to\Phi(\gamma)\to\beta)\to\alpha\to\beta)_{\xi(\alpha:=A_\xi)}$,

such that A_ξ, In_ξ and R_ξ only depend on $\xi\lceil FTV(\Phi)$ and

$$\mathsf{ev}\circ\langle\mathsf{ev}\circ\langle\llbracket z\beta\rrbracket_{\xi(\alpha:=A_\xi)}\circ\langle\mathsf{id},\mathsf{R}_\xi\rangle,g\rangle,\mathsf{ev}\circ\langle\mathsf{In}_\xi,b\rangle\rangle =$$

$$\mathsf{ev}\circ\langle\mathsf{ev}\circ\langle\mathsf{ev}\circ\langle\llbracket y\alpha\rrbracket_{\xi(\alpha=A_\xi)}\circ\langle k^{-}(\mathsf{id}),g\rangle,\mathsf{id}\rangle,\mathsf{ev}\circ\langle\llbracket z\beta\rrbracket_{\xi(\alpha:=A_\xi)}\circ\langle\mathsf{id},\mathsf{R}_\xi\rangle,g\rangle\rangle,b\rangle$$

for any $g:1\to\{\!|\forall\gamma.(\gamma{\to}\alpha){\to}(\gamma{\to}\beta){\to}\Phi(\gamma){\to}\beta|\!\}_\xi$, $b:1\to A_\xi$ and ξ with $\xi(\alpha)=A_\xi$.

2. *A K-model for polymorphic lambda calculus M is a model for F_{COREC} if for every positive type scheme Φ and object valuation ξ, there are*

   (a) *an object A_ξ,*

   (b) *a morphism $\mathsf{Out}_\xi:1\to\{\!|\Phi(\alpha)|\!\}^{A_\xi}_{\xi(\alpha:=A_\xi)}$,*

   (c) *a morphism $\Omega_\xi:1\to\{\!|\forall\beta.(\forall\gamma.(\alpha{\to}\gamma){\to}(\beta{\to}\gamma){\to}\beta{\to}\Phi(\gamma)){\to}\beta{\to}\alpha|\!\}_{\xi(\alpha=A_\xi)}$,*

   such that A_ξ, Out_ξ and Ω_ξ only depend on $\xi\lceil FTV(\Phi)$ and

$$\mathsf{ev}\circ\langle\mathsf{Out}_\xi,\mathsf{ev}\circ\langle\mathsf{ev}\circ\langle\llbracket z\beta\rrbracket_{\xi(\alpha:=A_\xi)}\circ\langle\mathsf{id},\Omega_\xi\rangle,g\rangle,b\rangle\rangle =$$

$$\mathsf{ev}\circ\langle\mathsf{ev}\circ\langle\mathsf{ev}\circ\langle\llbracket y\alpha\rrbracket_{\xi(\alpha=A_\xi)}\circ\langle\mathsf{id},g\rangle,\mathsf{id}\rangle,\mathsf{ev}\circ\langle\llbracket z\beta\rrbracket_{\xi(\alpha:=A_\xi)}\circ\langle\mathsf{id},\Omega_\xi\rangle,g\rangle\rangle,b\rangle$$

   for any $g:1\to\{\!|\forall\gamma.(\alpha{\to}\gamma){\to}(\beta{\to}\gamma){\to}\beta{\to}\Phi(\gamma)|\!\}_\xi$ and $b:1\to\xi(\beta)$.

Mendlers system has all syntactic recursive algebras and syntactic corecursive coalgebras.

Proposition 6.6 *For every positive type scheme $\Phi(\alpha)$*

1. *F_{REC} has a syntactic recursive Φ-algebra.*

2. *F_{COREC} has a syntactic corecursive Φ-coalgebra.*

Proof 1. Take $\sigma_0=\mu\alpha.\Phi(\alpha)$, $M_0=\mathbf{In}$ and $\mathrm{Rec}=\lambda\beta.\lambda g{:}\Phi(\beta\times\sigma_0){\to}\beta.\mathrm{R}\beta(\lambda\alpha.\lambda f{:}\alpha{\to}\sigma_0.\lambda k{:}\alpha{\to}\beta.g\circ\Phi(<k,f>))$.

2. Take $\sigma_1=\nu\alpha.\Phi(\alpha)$, $M_1=\mathbf{Out}$ and $\mathrm{Corec}=\lambda\beta.\lambda g{:}\beta{\to}\Phi(\beta{+}\sigma_1).\Omega\beta(\lambda\alpha.\lambda f{:}\sigma_1{\to}\alpha.\lambda k{:}\beta{\to}\alpha.\Phi([k,f])\circ g)$.

This Proposition implies that if one also adds the stronger equality rules for $\times$ and $+$ (of 6.1) to the syntax of F_{REC} and F_{COREC}, then in both systems one has all syntactic recursive algebras and all syntactic corecursive coalgebras. From that we easily obtain the following.

Corollary 6.7 *Let F be an expressible functor in polymorphic lambda calculus. Then all K-models for either F_{REC} or F_{COREC} have an expressible recursive F-algebra and an expressible corecursive F-coalgebra.*

In view of Definition 3.3, also Theorem 5.5 can be formulated syntactically as follows.

Proposition 6.8 *Suppose we work in an extension of polymorphic lambda calculus in which we have a notion of product and coproduct for which the stronger rules of 6.1 are valid. Then the following are equivalent.*

1. *There is a syntactic recursive Φ-algebra for every positive type scheme Φ.*

2. *There is a syntactic corecursive Φ-coalgebra for every positive type scheme Φ.*

The proof is by defining a syntactic recursive Φ-algebra in terms of a suitable syntactic corecursive Θ-coalgebra and vice versa, just as is done semantically in the proof of Theorem 5.5 and syntactically in the proof of Proposition 6.2. In fact it is much easier then in 5.5 to check everything here.

It doesn't seem possible to define the μ-types in terms of the ν-types in $F_{(CO)REC}$, nor to define the system F_{COREC} in a system with syntactic (co)recursive (co)algebras, for example in the system $F^{\times +}_{corec}$. When one attempts to do so it becomes clear what is needed to obtain a result like Theorem 5.5 or Proposition 6.2 for $F_{(CO)REC}$, or to obtain the reverse result of 6.6 (defining the μ-types (ν-types) of F_{COREC} in terms of syntactic (co)recursive (co)algebras.)

First one needs the extra equalities for $\times$ and $+$, as given by the reduction rules in Definition 6.1. But more imporantly, what seems to be necessary is that for all $f : \sigma \to \alpha$ and $g : \forall \beta.(\beta \to \sigma) \to (\beta \to \alpha) \to \Phi(\beta) \to \alpha$,

$$g\sigma(\mathrm{id}_\sigma)f = g(\tau \times \sigma)\mathrm{snd}\,\mathrm{fst} \circ \Phi(<f, \mathrm{id}_\sigma>)$$

and for all $f : \alpha \to \sigma$ and $g : \forall \beta.(\sigma \to \beta) \to (\alpha \to \beta) \to \alpha \to \Phi(\beta)$,

$$g\sigma(\mathrm{id}_\sigma)f = \Phi([f, \mathrm{id}_\sigma]) \circ g(\tau + \sigma)\mathrm{inr}\,\mathrm{inl}$$

In the syntax this is certainly not valid in general, but there is a class of models of polymorphic lambda calculus in which these equations are valid, namely those in which polymorphic terms of type $\forall \alpha.\Psi_1(\alpha) \to \Psi_2(\alpha)$ are (or maybe better 'act as') *dinatural transformations* (where Ψ_1 and Ψ_2 are type schemes, not necessarily positive.) We shall not go into this matter here, as this would require us to introduce a totally new topic; for more on dinatural transformations and their interest in polymorphic lambda calculus, see [Bainbridge et al. 1990]. Here we just want to mention the fact that if M is a K-model of polymorphic lambda calculus in which polymorphic terms act as dinatural transformations, then M is a model for F_{REC} if and only if M is a model for F_{COREC}.

7 The Calculus of Inductive Definitions

The formalisations of inductive and coinductive types we've been looking at so far are done in a *calculus*, a formal system for building terms, but not proofs. We now want to look at a type system in which one can also do logic to see how the (co)inductive types (should) behave there. The first candidate system to study is of course the Calculus of Constructions (CC) (see [Coquand and Huet 1988]), because it already includes all the inductive and coinductive types via the representations in system F, and it also includes a kind of higher order predicate logic. Now, besides that we can define the (co)iterative and (co)recursive functions on the (co)inductive types there are two extra requirements for (co)inductive types in a logical system. First, one would like the induction principle to be *provable*, that is e.g. for natural numbers, $(P0 \,\&\, \forall y \in \mathbf{N}(Py \to P(Sy))) \to (\forall x \in \mathbf{N}(Px))$ is provable for any predicate P on $\mathbf{N}$. Second, one would like to prove that the inductive types are not trivial, that is e.g. for the natural numbers, $0 \neq S0$ is provable. However, in CC the induction principle is not provable: $\mathbf{N}$ is just the weakly initial algebra of polymorphic Church numerals, as in system F, and one would have to relativise all formulas about $\mathbf{N}$ to the set of inductive natural numbers. Also $0 \neq S0$ is not provable because the equality in the system (Leibniz' equality) is too weak: From $0 = S0$ alone it's not possible to obtain a contradiction. To meet these requirements, the Calculus of Inductive Definitions has been defined ([Coquand and Mohring 1990]) and implemented ([Dowek e.a. 1991]). This system is set up in a similar way as $F^{\times +}_{(co)rec}$ and $F_{(CO)REC}$:There is a scheme for introducing new types, which come together with new constants and reduction rules. A fundamental extension is that now a proof by induction is done by the same scheme as a recursive function. Applying the techniques that have been used in the previous sections, the coinductive types can be defined in terms of the inductive ones. Before giving the syntax we want to turn to category.

Suppose C is a category in which the Calculus of Constructions can be interpreted in such a way that small types become objects. Suppose C also interprets small Σ types, i.e. the calculus with $\Sigma x{:}A.B(x){:}\mathrm{Prop}$ for $A{:}\mathrm{Prop}$, $B(x){:}\mathrm{Prop}(x{:}A)$. One can in fact just think of the term model for the Calculus of Constructions with strong small Σ types. The Σ types are used to interpret proofs by induction in terms of iteration, just as products are used to interpret recursion in terms of iteration. We first treat the example for Nat. In the following we shall be very informal with categorical notations, not distinguishing for example between $f{:}\Pi x{:}\mathrm{Nat}.Px{\to}P(Sx)$ and $x{:}\mathrm{Nat}, y{:}Px \vdash f(x,y){:}P(Sx)$.

Example 7.1 *Let* Nat *be the polymorphic Church numerals in the Calculus of Constructions. (In fact any weakly initial $\lambda X.1{+}X$-algebra will do.) and $P{:}\mathrm{Nat}{\to}Prop$. Then for $f_0{:}P0$ and $f_1{:}\Pi x{:}\mathrm{Nat}.Px{\to}P(Sx)$ we have that there's a term h for which the diagram*

$$\begin{tikzcd}
1 + \mathrm{Nat} \arrow[r, "{[\mathsf{Z}, \mathsf{S}]}"] \arrow[d, "\mathsf{id} + h"'] \arrow[dr, phantom, "="] & \mathrm{Nat} \arrow[d, "h"] \\
1 + (\Sigma x{:}\mathrm{Nat}.Px) \arrow[r, "{\langle [\mathsf{Z}, S \circ \pi_1], [f_0, \lambda z.f_1(\pi_1 z)(\pi_2 z)] \rangle}"'] & \Sigma x{:}\mathrm{Nat}.Px
\end{tikzcd}$$

commutes. Now, we would like to write $h = \langle h_1, h_2 \rangle$ and $h_1 = \mathsf{id}$ (which equations hold if Nat *is initial), because then we have*

$$h_2 : \Pi x{:}\mathrm{Nat}.Px.$$

To establish this we do not really need Nat *to be an initial algebra; it suffices if* Nat *is recursive. (Recall that a recursive $\lambda X.1 + X$-algebra arose by requiring that in the weak initiality diagram, $h_1 = \mathsf{id}$.)*

The general pattern is now as follows. Let T be a *strictly positive operator* from Prop to Prop, that is $X \notin FV(T(X))$ or $T(X) = \Pi\vec{y}{:}\vec{Y}.X$ with $X \notin FV(\vec{Y})$. We assume that T is in the original Calculus of Constructions, so it does not contain Σ-types. Note that such a T preserves composition (and if we would add the η-rule also identities.) The pair (A, f) is a recursive T-algebra in CC^{Σ} if

1. $A{:}\mathrm{Prop}$, $\mathbf{intro}{:}TA{\to}A$,
2. for any $P{:}(A{\to}Prop)$, if $g{:}(\Pi x{:}TA.T(P)x{\to}P(\mathbf{intro}x))$ there is a term $h{:}(\Pi x{:}A.Px)$ such that the diagram

$$\begin{tikzcd}
TA \arrow[r, "\mathbf{intro}"] \arrow[d, "T(\langle \mathsf{id}, h \rangle)"'] \arrow[dr, phantom, "="] & A \arrow[d, "\langle \mathsf{id}, h \rangle"] \\
T(\Sigma x{:}A.Px) \arrow[r, "{\langle \mathbf{intro} \circ T(\pi_1), \lambda z.g(T(\pi_1)z)(T(\pi_2)z) \rangle}"'] & \Sigma x{:}A.Px
\end{tikzcd}$$

   commutes. Here the lifting of $P{:}X{\to}\mathrm{Prop}$ to a predicate $T(P){:}TX{\to}\mathrm{Prop}$ is defined by taking for $TX = \Pi\vec{y}{:}\vec{Y}.X$, $T(P) = \lambda z{:}TX.\Pi\vec{y}{:}\vec{Y}.P(z\vec{y})$. The lifting of a dependent function $h{:}\Pi z{:}X.Pz$, $T(h)$ is then defined as a function of type $\Pi z{:}TX.T(P)z$ by taking for $TX = \Pi\vec{y}{:}\vec{Y}.X$, $T(h) = \lambda z{:}TX.\lambda\vec{y}{:}\vec{Y}.h(z\vec{y})$.

The equality rules that arise from the commutativity of the diagram above are

$$\begin{aligned}
\mathsf{id} \circ \mathbf{intro} &= \mathbf{intro} \circ T(\pi_1) \circ T(\langle \mathsf{id}, h \rangle), \\
h \circ \mathbf{intro} &= \lambda y.g(T(\pi_1)y)(T(\pi_2)y) \circ T(\langle \mathsf{id}, h \rangle).
\end{aligned}$$

The first equation is always valid and the second can be rewritten to

$$h \circ \mathbf{intro} = \lambda y.gy(T(h)y).$$

(We are only concerned with strictly positive operators, because for operators T for which $T(X)$ does not contain X, or $T(X)$ contains X, but not strictly positively, these definitions do not make sense. For example if T is a positive operator and P happens to be a constant function (so h is not really type dependent function, but of type $A{\to}B$), then the two possible definitions of $T(h)$ are not the same.)

With these intuitions in mind, we now want to give the syntax of the calculus of inductive definitions. In fact this is one possible construction, because there are different ones depending on what underlying type system we start with. Our objective here is to get a typed lambda calculus which unifies constructive higher order predicate logic with a calculus of inductive types, so at this point we don't bother about program extraction. Also we want to avoid elements, depending on proofs and sets depending on elements to keep the system transparent. The system defined below is a subsystem of the one implemented as Coq. (See [Dowek e.a. 1991])

Definition 7.2 *The Pure Type System λHOPP (Higher Order Predicate logic with Polymorphic sets) is defined by the following specification.*

$$\begin{array}{rcl}\mathcal{S} & = & \mathit{Set}, \mathit{Prop}, \mathit{Type}^s, \mathit{Type}^p,\\ \mathcal{A} & = & \mathit{Set} : \mathit{Type}^s, \mathit{Prop} : \mathit{Type}^p,\\ \mathcal{R} & = & (\mathit{Set}, \mathit{Set}), (\mathit{Type}^s, \mathit{Set}), (\mathit{Set}, \mathit{Type}^p), (\mathit{Type}^p, \mathit{Type}^p),\\ & & (\mathit{Prop}, \mathit{Prop}), (\mathit{Set}, \mathit{Prop}), (\mathit{Type}^p, \mathit{Prop}).\end{array}$$

The part concerning Set and Type^s is system F; the rules (Set ,Type^p) and (Type^p ,Type^p) extend this with a higher order language and the rules with Prop give the logic (implication, first order quantification and higher order quantification.) We could also have started with sytem Fω in stead of F, but this extension is not necessary for what we want to do.

Definition 7.3 ([Coquand and Mohring 1990]) *1. For Φ a set, proposition or type in λHOPP and X a variable of the same sort, X* is strictly positive in *Φ if $X \notin FV(\Phi)$ or if $\Phi = \Pi\vec{y}{:}\vec{Y}.X$ with $X \notin FV(\vec{Y})$. (We view Φ as a type scheme and to stress that we sometimes write $\Phi(X)$.If there's no ambiguity w.r.t. the variable X, we just say that Φ is strictly positive.)*

2. *For $\Phi(X) = \Pi\vec{y}{:}\vec{Y}.X$ strictly positive and $P(x){:}s(x{:}B)$ we define* the lifting of P along Φ*, $\Phi[P](x){:}s(x{:}\Phi(B))$, by $\Phi[P](x) := \Pi\vec{y}{:}\vec{Y}.P(x\vec{y})$.)*

3. *For $\Phi(X) = \Pi\vec{y}{:}\vec{Y}.X$ strictly positive, $P(x){:}s(x{:}B)$ and $f{:}\Pi x{:}B.P(x)$, we define* the lifting of f along Φ*, $\Phi[f]{:}\Pi x{:}\Phi(B).\Phi[P](x)$ by $\Phi[f] := \lambda x{:}\Phi(B).\Pi\vec{y}{:}\vec{Y}.f(x\vec{y})$.)*

Definition 7.4 *1. The scheme $\Theta(X)$ of type Set is a* constructor *if $\Theta(X) = \Pi\vec{x}{:}\vec{\Phi(X)}.X$, with all $\Phi_i(X)$ strictly positive.*

2. *Let $\Theta{:}s$ be a constructor with $\Phi_{i_1}, \ldots, \Phi_{i_p}$ the strictly positive operators in Θ in which X occurs. For $P(z){:}s'(z{:}A)$ we define $\Theta[P](z){:}s'(z{:}\Theta(A))$ by*

$$\Theta[P](z) := \Pi\vec{x} : \vec{\Phi(A)}.\Phi_{i_1}[P](x_{i_1}){\to}\ldots{\to}\Phi_{i_p}[P](x_{i_p}){\to}P(z\vec{x}).$$

Definition 7.5 *The* inductive type scheme *for forming new propositions, sets and types is the following. Let $\Theta_1(X), \ldots, \Theta_n(X)$ be constructors, then we have*

1. *a new set $\mu X.[\Theta_1, \ldots, \Theta_n]$, usually abbreviated to μ,*
2. *new constants $\mathbf{intro}_i : \Theta_i(\mu)$ for every $1 \leq i \leq n$,*
3. *a new derivation rule*

$$\frac{\Gamma, z{:}\mu \vdash P(z) : s \quad \Gamma \vdash f_1 : \Theta_1[P](\mathbf{intro}_1) \ldots \Gamma \vdash f_n : \Theta_n[P](\mathbf{intro}_n)}{\Gamma \vdash \mathrm{Rec} f_1 \ldots f_n : (\Pi z{:}\mu.P(z))}$$

4. a new reduction rule

$$\mathrm{Rec}f_1 \ldots f_n(\mathbf{intro}_i t_1 \ldots t_m) \longrightarrow f_i t_1 \ldots t_m(\Phi_{i_1}[R]t_{i_1}) \ldots (\Phi_{i_p}[R]t_{i_p}),$$

where $\Theta_i = \Pi\vec{x}{:}\Phi(\vec{X}).X$ and $\Phi_{i_1} \ldots \Phi_{i_p}$ are those Φ in which X really occurs. $\mathrm{Rec}f_1 \ldots f_n$ is abbreviated to R.

We have to say what possible sorts s we allow in the rules. In the derivation rule we allow $s \in \{Set, Prop\}$, for $\mu{:}Set$. The type μ can be formed if $\Theta(X) : Set$ for $X{:}Set$.

Definition 7.6 *$\lambda HOPPI$ is the system $\lambda HOPP$ with the inductive type scheme as given above.*

Example 7.7 *The inductive type of natural numbers is now*

$$\mathrm{Nat} := \mu X.[X, X{\to}X],$$

which comes with the constants $0{:}\mathrm{Nat}$ and $S{:}\mathrm{Nat}{\to}\mathrm{Nat}$, the derivation rule

$$\frac{\Gamma, x{:}\mathrm{Nat} \vdash P(x) : \mathit{Prop/Set} \quad \Gamma \vdash f_1 : P(0) \quad \Gamma \vdash f_2 : \Pi x{:}\mathrm{Nat}.P(x){\to}P(Sx)}{\Gamma \vdash \mathrm{Rec}f_1f_2 : \Pi x{:}\mathrm{Nat}.P(x)}$$

and the reduction rules

$$\mathrm{Rec}f_1f_2 0 \longrightarrow f_1,\ \mathrm{Rec}f_1f_2(Sx) \longrightarrow f_2x(\mathrm{Rec}f_1f_2x).$$

Another example is

$$A + B := \mu X.[A{\to}X, B{\to}X],$$

which comes with $\mathbf{intro}_1{:}A{\to}A{+}B$, $\mathbf{intro}_2{:}B{\to}A{+}B$ and for $f_1{:}\Pi x{:}A.P(\mathbf{intro}_1x)$, $f_2{:}\Pi x{:}B.P(\mathbf{intro}_2x)$, $\mathrm{Rec}f_1f_2{:}\Pi x{:}A + B.P(x)$

We can not (as in [Coquand and Mohring 1990]) define the type $\Sigma x{:}A.Qx := \mu X.[\Pi x{:}A.Qx{\to}X]$ (a strong sigma type), for $A{:}\mathrm{Set}$, $Q{:}A{\to}\mathrm{Prop}$, because we haven't allowed $Qx{\to}X$ to be formed. This would require the rule (Prop, Set), but then we get elements of sets depending on proofs, which we don't want at this point. We *can* define the sigma type $\Sigma Y{:}\mathrm{Set}.B(Y)$ by $\mu X.[\Pi Y{:}\mathrm{Set}.B(Y){\to}X]$, which has only a second projection.

We now want to show how to define coinductive types with corecursion using the inductive types scheme. Applying the translations of coinductive types in terms of inductive types of the previous sections, the following seems reasonable.

Definition 7.8 *For Φ a strictly positive operator, we define the coinductive type from Φ by*

$$\nu Y.\Phi(Y) := \mu X.[\Pi Y{:}\mathit{Set}.(Y{\to}\Phi(Y + X)){\to}Y{\to}X].$$

($\nu Y.\Phi(Y)$ is usually abbreviated by ν.)

Now an inductively defined set $Y + X$ occurs in the constructor Θ, so we have to say what $\Theta[P]$ is in this new case. We are done if we define $\Phi[P] : \Phi(A){\to}\mathrm{Set/Prop}$ for $\Phi X = Y + X$. To do that we extend the underlying system λHOPP with the rule $(\mathrm{Set}, \mathrm{Type}^s)$ and we allow, in the derivation rule for the recursive function on $Y + X$ (for $Y, X{:}\mathrm{Set}$) s to be Type^p or Type^s. Now, if $P{:}A{\to}\mathrm{Prop}$ take $\Phi[P] := \mathrm{Rec}(\lambda z{:}Y.\top)P$, which is formed by

$$\frac{\Gamma, t{:}Y + A \vdash \mathrm{Prop} : \mathrm{Type}^p \quad \Gamma \vdash \lambda y{:}Y.\top : Y{\to}\mathrm{Prop} \quad \Gamma \vdash P : A{\to}\mathrm{Prop}}{\Gamma \vdash \mathrm{Rec}(\lambda z{:}Y.\top)P : Y + A{\to}\mathrm{Prop}}$$

Here $\top : \mathrm{Prop}$ denotes $\Pi\alpha : \mathrm{Prop}.\alpha{\to}\alpha$. If $P{:}A{\to}\mathrm{Set}$, take $\Phi[P] := \mathrm{Rec}(\lambda z{:}Y.1)P$, which is formed by

$$\frac{\Gamma, t{:}Y + A \vdash \text{Set} : \text{Type}^s \quad \Gamma \vdash \lambda y{:}Y.1 : Y{\to}\text{Set} \quad \Gamma \vdash P : A{\to}\text{Set}}{\Gamma \vdash \text{Rec}(\lambda z{:}Y.1)P : Y + A{\to}\text{Set}}$$

Here 1:Set denotes the set $\mu X[X]$.

For $\Phi(X)$ a strictly positive operator, write $\Phi'_Y(X)$ for $Y{\to}\Phi(Y + X)$. Then the coinductive type $\nu X.\Phi(X) := \mu X.[\Pi Y{:}\text{Set}.\Phi'_Y(X){\to}Y{\to}X]$ comes together with

$$\textbf{intro} : \Pi Y : \text{Set}.\Phi'_Y(\nu){\to}Y{\to}\nu$$

and the rule

$$\frac{\Gamma, z{:}\nu \vdash P(z) : \text{Prop/Set} \quad \Gamma \vdash f : \Pi Y{:}\text{Set}.\Pi y_1{:}\Phi'_Y(\nu).\Pi y_2{:}Y.\Phi'_Y[P]y_1{\to}P(\textbf{intro}Y y_1 y_2)}{\Gamma \vdash \text{Rec}f : \Pi z{:}\nu.P(z)}$$

We treat the example for Stream to show that this type really enjoys corecursion.

Example 7.9 *Define*

$$\text{Stream} := \mu X.[\Pi Y{:}\mathit{Set}.(Y{\to}\text{Nat}){\to}(Y{\to}Y + X){\to}Y{\to}X].$$

Then

$$\textbf{intro} : \Pi Y{:}\mathit{Set}.(Y{\to}\text{Nat}){\to}(Y{\to}Y + \text{Stream}){\to}Y{\to}\text{Stream}$$

and

$$\frac{\Gamma, z{:}\text{Stream} \vdash P{:}s \quad \Gamma \vdash f : \Pi Y.\Pi y_1{:}Y{\to}\text{Nat}.\Pi y_2{:}Y{\to}Y + \text{Stream}.\Pi y_3{:}Y.(\Phi'_Y[P](y_2)){\to}P(\textbf{intro}Y y_1 y_2 y_3)}{\Gamma \vdash \text{Rec}f : \Pi z{:}\text{Stream}.P,}$$

where s is Set or Prop, $\Phi'_Y[P](y_2)$ is defined as $\Pi y{:}Y.\textbf{rec}(\lambda z.I)P(y_2 y)$ with **rec** *the recursor for the inductive type $Y + \text{Stream}$ and I is 1 or $\top$, depending on s. The reduction rule is*

$$\text{Rec}f(\textbf{intro}Y y_1 y_2 y_3) \longrightarrow fY y_1 y_2 y_3(\Phi'_Y[\text{Rec}f]y_2),$$

where $\Phi'_Y[\text{Rec}f] := \lambda z{:}Y{\to}Y{+}\text{Stream}.\lambda y{:}Y.\textbf{rec}(\lambda x.!)(\text{Rec}f)(zy)$ (The term $!$ is the unique term of either $1{:}\mathit{Set}$ or $\top{:}\mathit{Prop}$.) Define now

$$\begin{array}{rcl} H & := & \text{Rec}(\lambda Y y_1 y_2 y_3 y.y_1 y_3) : \text{Stream}{\to}\text{Nat}, \\ T & := & \text{Rec}(\lambda Y y_1 y_2 y_3 y.\big[\textbf{intro}Y y_1 y_2, \text{id}\big](y_2 y_3)) : \text{Stream}{\to}\text{Stream}, \end{array}$$

which satisfy $T(\textbf{intro}Y y_1 y_2 y_3) \twoheadrightarrow \big[\textbf{intro}Y y_1 y_2, \text{id}\big](y_2 y_3)$ and $H(\textbf{intro}Y y_1 y_2 y_3) \twoheadrightarrow y_1 y_3$, the equation rules for corecursion. The function ZeroH*, which replaces the head of a stream by* 0 *is defined by* $\text{ZeroH} := \textbf{intro}\,\text{Stream}(\lambda s.0)(\lambda s.\text{inr}(Ts))$.

With the previous example, we can also construct proofs of $\Pi x{:}\text{Stream}.Pz$ (if $P{:}\text{Stream}{\to}\text{Prop}$.) It is not clear to what can be done with this feature, as we don't have any examples of proofs of propositions which essentially use this scheme. A $y{:}\Phi'_Y[P](y_2)$ gives for $y_2 = \text{inl} \circ h$ a proof of $\Pi y'{:}Y.\top$ and for $y_2 = \text{inr} \circ h$ a proof of $\Pi y'{:}Y.P(hy')$.

A proof of $\Pi Y.\Pi y_1{:}Y{\to}\text{Nat}.\Pi y_2{:}Y{\to}Y{+}\text{Stream}.\Pi y_3{:}Y.(\Phi'_Y[P](y_2)){\to}P(\textbf{intro}Y y_1 y_2 y_3)$ is saying that P holds for all coiterative streams and, for any $y_1{:}Y{\to}\text{Nat}$, $h{:}Y{\to}\text{Stream}$, $y_3{:}Y$ if P holds for any possible stream hy', then P holds for $\textbf{intro}Y y_1(\text{inr} \circ h)y_3$. This seems to make sense, but, as already remarked, we don't have any examples of the use of this.

Discussion

We have described general categorical notions of recursion and corecursion, which seem to capture quite well our intuitive understanding of the concepts. It would be interesting to see which models of (polymorphic) typed lambda calculus, that do not have all initial algebras and terminal coalgebras, do have recursive algebras and corecursive coalgebras. It has been shown that in the realm of polymorphism, recursion and corecursion can be translated in each other, which raises the question whether there is one more general concept from which both recursion and corecursion are derivatives (using polymorphism.) In the last section we have applied this to a system in which also proofs by induction are done using the recursion scheme. It's not clear whether the dualization gives us a scheme for proofs 'by coinduction' and if so, whether this really yields something new.

Acknowledgements

We would like to thank Bart Jacobs for the numerous discussions on the subject, Christine Paulin-Mohring for explaining the Calculus of Inductive Definitions, Nax Mendler for pointing out his work on the subject and Thorsten Altenkirch pointing out the work of G. Wraith.

References

[Bainbridge et al. 1990] E.S. Bainbridge, P.J. Freyd, A. Scedrov and P.J. Scott, Functorial polymorphism, *Theor. Comp. Sc.*, 70, pp 35-64.

[Böhm and Berarducci 1985] C. Böhm and A. Berarducci, Automatic synthesis of typed Λ-programs on term algebras *Theor. Comput. Science*, 39, pp 135-154.

[Coquand and Huet 1988] Th. Coquand and G. Huet, The calculus of constructions, *Information and Computation*, 76, pp 95-120.

[Coquand and Mohring 1990] Inductively defined types, In P. Matrin-Löf and G. Mints editors. *COLOG-88 : International conference on computer logic, LNCS 417.*

[Dowek e.a. 1991] G. Dowek, A. Felty, H. Herbelin, G. Huet, C. Paulin-Mohring, B. Werner, The Coq proof assistant version 5.6, user's guide. INRIA Rocquencourt - CNRS ENS Lyon.

[Girard 1972] J.Y. Girard, Interprétation foctionelle et élimination des coupures dans l'arithmétique d'ordre supérieur. Ph.D. thesis, Université Paris VII.

[Girard et al. 1989] J.Y. Girard, Y. Lafont and P. Taylor, *Proofs and types*, Camb. Tracts in Theoretical Computer Science 7, Cambridge University Press.

[Hagino 1987a] T. Hagino, A categorical programming language, Ph. D. thesis, University of Edinburgh.

[Hagino 1987b] T. Hagino, A typed lambda calculus with categorical type constructions. In D.H. Pitt, A. Poigné and D.E. Rydeheard, editors. *Category Theory and Computer Science, LNCS 283* pp 140-157.

[Hayashi 1985] S. Hayashi, Adjunction of semifunctors: categorical structures in nonextensional lambda calculus. *Theor. Comp. Sc.41*, pp 95-104.

[Kleene 1936] S.C. Kleene, λ-definability and recursiveness. *Duke Math. J.* 2, pp 340-353.

[Lambek 1968] J. Lambek, A fixed point theorem for complete categories. *Mathematisches Zeitschrift 103* pp 151-161.

[Leivant 1989] D. Leivant, Contracting proofs to programs. In P. Odifreddi, editor. *Logic in Computer Science*, Academic Press, pp 279-327.

[Mendler 1987] N.P. Mendler, Inductive types and type constraints in second-order lambda calculus. *Proceedings of the Second Symposium of Logic in Computer Science.* Ithaca, N.Y., IEEE, pp 30-36.

[Mendler 1991] N.P. Mendler, Predicative type universes and primitive recursion. *Proceedings of the Sixth Annual IEEE Symposium on Logic in Computer Science.* Amsterdam, The Netherlands, IEEE, pp 173-184

[Reynolds 1974] J.C. Reynolds, Towards a theory of type structure. *Proceedings, Colloque sur la Programmation LNCS 19* pp 408-425.

[Reynolds 1984] J.C. Reynolds, Polymorphism is not set-theoretic. In G. Kahn, D.B. MacQueen and G.D. Plotkin, editors. *Semantics of Data Types, LNCS 173* pp145-156.

[Reynolds and Plotkin 1990] J.C. Reynolds and G.D. Plotkin, On functors expressible in the polymorphic lambda calculus. In G. Huet, editor. *Logical Foundations of Functional Programming*, In 'The UT Year of Programming Series', Austin, Texas, pp 127-152.

[Wraith 1989] G.C. Wraith, A note on categorical datatypes In D.H. Pitt, A. Poigné and D.E. Rydeheard, editors. *Category Theory and Computer Science, LNCS 389* pp 118-127.