

# Generalized Coiteration Schemata<sup>1</sup>

Daniela Cancila, Furio Honsell, Marina Lenisa

*Dipartimento di Matematica e Informatica, Università di Udine  
Via delle Scienze 206, 33100 Udine, Italy. [cancila,honsell,lenisa@dimi.uniud.it](mailto:cancila,honsell,lenisa@dimi.uniud.it)*

---

## Abstract

*Coiterative functions* can be explained categorically as *final coalgebraic morphisms*, once coinductive types are viewed as final coalgebras. However, the coiteration schema which arises in this way is too rigid to accommodate directly many interesting classes of circular specifications. In this paper, building on the notion of *T-coiteration* introduced by the third author and capitalizing on recent work on *bialgebras* by Turi-Plotkin and Bartels, we introduce and illustrate various generalized coiteration patterns. First we show that, by choosing the appropriate monad  $T$ ,  $T$ -coiteration captures naturally a wide range of coiteration schemata, such as the duals of *primitive recursion* and *course-of-value iteration*, and *mutual coiteration*. Then we show that, in the more structured categorical setting of bialgebras,  $T$ -coiteration captures *guarded coiterations* schemata, i.e. specifications where recursive calls appear guarded by predefined algebraic operations.

**Keywords:** coinductive datatype, categorical semantics, coalgebra, bialgebra, coiteration schema, guarded specification.

---

## Introduction

*F-coalgebras*, for  $F$  endofunctor on a category  $\mathcal{C}$ , offer a uniform categorical account of the behaviour of *dynamical systems* and various kinds of *circular* and *infinite* objects [Acz88, JR96, BM96, Rut00]. According to this paradigm, *universal systems* and *coinductive datatypes* are modeled as *final coalgebras*, *bisimulations* are modeled as suitable coalgebras over binary relations on states, called *coalgebraic bisimulations*, and *coiterative functions* into coinductive datatypes are explained as the unique *coalgebraic morphisms* into the final coalgebra.

This approach has been very fruitful and it originated a new area of semantics called *final semantics*, whereby the operational behaviour of a process algebra is captured as a coalgebra over the syntax and the interpretation function, inducing the intended observational equivalence, is construed as a final mapping into a final coalgebra of suitable denotations, see e.g. [Acz93, RT94, Len96, JR96].

---

<sup>1</sup> Research supported by the MIUR Project COFIN 2001013518 COMETA.

However, the coiteration schema which arises from coalgebraic morphisms is very rigid. Natural circular specifications of functions need a lot of pre-processing before they can be cast into this pattern. To overcome this drawback, various authors [Gim94,Pav98,Len99,UV99,Bar01,UVP01] have recently addressed the problem of generalizing the basic categorical setting to accommodate directly more expressive classes of circular specifications, e.g. duals of *primitive recursion* and *course-of-value iteration*, *mutual coiteration* etc. To this end, the third author of this paper introduced in [Len99] the schema of  $T$ -coiteration for  $T$  pointed endofunctor.

A similar limitation arises in connection with bisimulations. In practice, many notions of *bisimulations up-to* are in use, see e.g. [San98,Len99], offering often the advantage of being finite even when the corresponding standard bisimulations are not. These cannot be directly expressed in terms of basic coalgebraic bisimulations, but require appropriate generalizations [Len99,LPW00,Bar01].

Often dynamical systems come equipped with *algebraic operations* w.r.t. which the intended behavioural semantics is a *congruence*, the standard case in point being syntactic constructors on terms in process algebras. Plain unstructured coalgebras are too weak to allow for principled proofs of congruence. An appropriate setting in which to discuss these issues is that of *bialgebras* and *structured coalgebras* [TP97,CHM01], i.e. suitable objects which have both an algebra and a coalgebra structure. In this setting, coiterative morphisms appear also as *algebra homomorphisms*, and the equivalences induced by coiterative morphisms are *congruences* w.r.t. the operations on the algebra.

In this paper, building on the notion of  $T$ -coiteration and capitalizing on recent work on bialgebras, we introduce and illustrate the expressivity of various generalized categorical coiteration patterns.

First we show that the  $T$ -coiteration schema captures naturally a wide range of patterns, such as the duals of *primitive recursion* and *course-of-value iteration*, and *mutual coiteration*. Hence, in particular  $T$ -coiteration is *extensionally universal*, i.e. it captures the graph of any morphism in the underlying category. Then we show that, in the more structured categorical setting of generalized  $\lambda$ -bialgebras, the  $T$ -coiteration schema captures *guarded coiterations*, i.e. specifications where recursive calls appear guarded by predefined algebraic operations. In this setting, a very interesting and general class of  $T$ -coiteration schemata is the one which arises when  $T$  is the *free monad* generated by the functor for which the guard is an algebra operation. For ease of presentation, many concrete examples are provided on the datatype of streams, including Hamming, Fibonacci, as well as some operations on dynamical systems.

The use of  $\lambda$ -bialgebras and distributive laws in dealing with generalized coiteration was first developed in [Bar01]. The present work stems from an alternative categorical account of the results in [Bar01] on guarded schemata. In our approach, we *actually* work in an ambient category of possibly generalized  $\lambda$ -bialgebras. We feel that our approach suggests a new perspective on the use of bialgebras in the context of generalized coiteration, whereby

guards in coiteration schemata are construed as *algebraic constructors* and *coiterative morphisms* appear also as *algebra homomorphisms*. Furthermore, we introduce a notion of *generalized distributive law*, which generalizes that in [Bar01].

We conclude the paper by hinting at a general language for mutual coinductive specifications which features all the schemata discussed in the paper. Its categorical semantics, based on  $T$ -coiteration, is *compositional*, i.e. the semantics of complex schemata is obtained by composing the monads used for dealing with the elementary component schemata separately.

This modular semantics suggests naturally *effective syntactic procedures* for validating *mutual circular specifications*. These can be used in interactive proof development environments, based on type theory, such as Coq, for extending the classes of coiteration schema currently accepted, [Coq94].

*Synopsis.* In Section 1, we introduce the relevant categorical notions. In Section 2, we present a collection of motivating examples of circular definitions of functions on streams. We discuss coiteration and its limitations in Section 3. The expressivity of  $T$ -coiteration is analyzed in Section 4. In Section 5 we discuss guarded specifications and illustrate the added value obtained by working in a suitable ambient category of bialgebras. Final remarks and directions for future work appear in Section 6.

*Acknowledgement.* The authors would like to thank F.Bartels for useful comments on a preliminary version of the paper.

## 1 Coalgebras and Bialgebras

In this section, we introduce various notions and results which are relevant to the categorical treatment of coinductive types. We work in a category  $\mathcal{C}$  which has finite products and coproducts.

**Definition 1.1** [ $G$ -algebra,  $F$ -coalgebra,  $\langle G, F \rangle$ -bialgebra] An *algebra for the functor*  $G : \mathcal{C} \rightarrow \mathcal{C}$  ( $G$ -*algebra*) is a pair  $(X, \beta_X)$ , where  $X$  is an object of  $\mathcal{C}$  (the *carrier* of the algebra) and  $\beta_X : GX \rightarrow X$  is an arrow in  $\mathcal{C}$  (the *operation* of the algebra). Dually, an  $F$ -*coalgebra* is a pair  $(X, \alpha_X)$ , where  $\alpha : X \rightarrow FX$ . A  $\langle G, F \rangle$ -*bialgebra* is a triple  $(X, \beta_X, \alpha_X)$ , where  $(X, \beta_X)$  is a  $G$ -algebra and  $(X, \alpha_X)$  is an  $F$ -coalgebra.

Algebra operations allow to *construct* elements of the carrier. Coalgebra operations, called *destructors* or *unfoldings*, yield information on the *states* of the dynamical system represented by the coalgebra. It is often the case, as we will see, that states in a dynamical system come equipped with an algebraic structure. Bialgebras account for this richer structure.

$G$ -algebras,  $F$ -coalgebras, and  $\langle G, F \rangle$ -bialgebras can all be endowed with a suitable structure of category by defining the notions of  $G$ -algebra,  $F$ -coalgebra,  $\langle G, F \rangle$ -bialgebra morphism, respectively, as follows:

**Definition 1.2** An arrow  $f : X \rightarrow Y$  is a  $G$ -*algebra morphism* from the  $G$ -algebra  $(X, \beta_X)$  to the  $G$ -algebra  $(Y, \beta_Y)$  if it makes diagram (1) commute.

An arrow  $f : X \rightarrow Y$  is an  $F$ -coalgebra morphism from the  $F$ -coalgebra  $(X, \alpha_X)$  to the  $F$ -coalgebra  $(Y, \alpha_Y)$  if it makes diagram (2) commute. An arrow  $f : X \rightarrow Y$  is a  $\langle G, F \rangle$ -bialgebra morphism from  $(X, \beta_X, \alpha_X)$  to  $(Y, \beta_Y, \alpha_Y)$ , if  $f$  makes both diagrams (1) and (2) commute.

$$\begin{array}{ccccc} GX & \xrightarrow{\beta_X} & X & \xrightarrow{\alpha_X} & FX \\ Gf \downarrow & (1) & \downarrow f & (2) & \downarrow Ff \\ GY & \xrightarrow{\beta_Y} & Y & \xrightarrow{\alpha_Y} & FY \end{array}$$

We will denote by  $Alg_G$ ,  $Coalg_F$ , and  $Bialg_{G,F}$  the categories of  $G$ -algebras,  $F$ -coalgebras, and  $\langle G, F \rangle$ -bialgebras, respectively.

Bialgebras, as defined above, combine the algebraic and coalgebraic structures *independently*. Much more interesting are those bialgebras where there is a tighter connection between the two structures, e.g. when  $F$  ( $G$  respectively) lifts to a (co)algebra functor, so that  $\alpha_X$  ( $\beta_X$  respectively) becomes a (co)algebra morphism. This notion of structured bialgebra has been very fruitful in the context of final semantics to streamline the proofs that bisimilarities are *congruences* w.r.t. suitable syntactical operators of the language. A very natural sufficient condition for the above to happen is the pentagonal law of  $\lambda$ -bialgebras, [TP97]. For a given bialgebra  $(X, \beta_X, \alpha_X)$ , such a law essentially allows to endow with a  $G$ -algebra structure  $FX$ , in such a way that  $\alpha_X$  is a  $G$ -algebra morphism between  $(X, \beta_X)$ , and the  $G$ -algebra with carrier  $FX$  thus defined. And dually.

We introduce a generalization of this condition, for  $F, G$  functors, called the *generalized pentagonal law*, which will be used in Section 5 for capturing generalized coiteration schemata. First we need the following definition:

**Definition 1.3** [Strict Coalgebra Functor] Let  $F, H : \mathcal{C} \rightarrow \mathcal{C}$ . A *strict coalgebra functor* is a functor  $\mathcal{F} : Coalg_F \rightarrow Coalg_H$  such that, for any  $F$ -coalgebra  $(X, \alpha_X)$ ,  $\mathcal{F}(X, \alpha_X) = (X, \theta_X)$ , for some  $\theta_X : X \rightarrow HX$ , and for any  $F$ -coalgebra morphism  $f : (X, \alpha_X) \rightarrow (Y, \alpha_Y)$ ,  $\mathcal{F}(f) = f$ .

**Definition 1.4** [Bialgebra for a (Generalized) Pentagonal Law] Let  $F, G, H : \mathcal{C} \rightarrow \mathcal{C}$ , let  $\mathcal{F} : Coalg_F \rightarrow Coalg_H$  be strict. A  $\lambda$ -bialgebra for a  $\mathcal{F}$ -generalized pentagonal law is a  $\langle G, F \rangle$ -bialgebra  $(X, \beta_X, \alpha_X)$  such that the following pentagonal diagram commutes:

$$\begin{array}{ccccc} GX & \xrightarrow{\beta_X} & X & \xrightarrow{\alpha_X} & FX \\ G(\theta_X) \downarrow & & & & \uparrow F(\beta_X) \\ G(HX) & \xrightarrow{\lambda_X} & FGX & & \end{array}$$

where  $\lambda$  is a  $\mathcal{F}$ -generalized distributive law of the functor  $G$  over the functor  $F$ , i.e.  $\lambda : GH \rightarrow FG$ , and  $\mathcal{F}(X, \alpha_X) = (X, \theta_X)$ .

The full subcategory of  $Bialg_{G,F}$  of  $\lambda$ -bialgebras is denoted by  $Bialg_\lambda$ .

Turi-Plotkin's original notion of  $\lambda$ -bialgebra, for the case of  $F, G$  functors, can be recovered by taking  $H \triangleq F$  and  $\mathcal{F} \triangleq Id_{Coalg_F}$ . For the case of  $F, G$

monad and comonad considered in [TP97], one can probably give a corresponding generalization of the distributive law, but this is out of the scope of the paper.

Interesting classes of generalized  $\lambda$ -bialgebras are the  $\lambda_I$ -bialgebras, which we define as follows:

**Definition 1.5** [ $\lambda_I$ -bialgebra] Let  $I \subseteq_{fin} Nat$ . A  $\lambda_I$ -bialgebra is a generalized  $\lambda$ -bialgebra for  $H \triangleq \prod_{i \in I} F^i$  and  $\mathcal{F} : Coalg_F \rightarrow Coalg_H$  defined by  $\mathcal{F}(X, \alpha_X) \triangleq (X, \langle (\alpha_X)_i \rangle)$ , where  $(\alpha_X)_i : X \rightarrow F^i(X)$  is defined by induction on  $i$  as follows:  $(\alpha_X)_0 \triangleq id_X$ ,  $(\alpha_X)_{i+1} \triangleq F^i(\alpha_X) \circ (\alpha_X)_i$ .

In particular, for  $I \subseteq \{0, 1\}$  we recover the class of generalized  $\lambda$ -bialgebras introduced in [Bar01].

Notice that the commutativity of the generalized pentagonal diagram of Definition 1.4 still implies that  $\beta_X$  is a coalgebra morphism from the  $F$ -coalgebra  $(GX, \lambda_X \circ G(\theta_X))$  to the  $F$ -coalgebra  $(X, \alpha_X)$ . However, the dual fails, in general.

The important property of  $\lambda$ -bialgebras is that *final coalgebras* in the underlying category of coalgebras “lift” *uniquely* to *final bialgebras* in  $Bialg_\lambda$ :

**Proposition 1.6** Let  $\lambda : GH \rightarrow FG$  be a  $\mathcal{F}$ -generalized distributive law of  $G$  over  $F$ . If  $(\Omega, \alpha_\Omega)$  is a final  $F$ -coalgebra, then there exists a unique  $\beta_\Omega : G\Omega \rightarrow \Omega$  such that  $(\Omega, \beta_\Omega, \alpha_\Omega)$  is a  $\lambda$ -bialgebra. Moreover,  $(\Omega, \beta_\Omega, \alpha_\Omega)$  is final in  $Bialg_\lambda$ .

Finally we recall the definition of *pointed endofunctor* and *monad*, and we introduce two distinguished monads which will be used in the sequel.

**Definition 1.7** [Pointed Endofunctor, Monad]

- A *pointed endofunctor* over a category  $\mathcal{C}$  is a pair  $\langle T, \eta \rangle$ , where  $T$  is an endofunctor on  $\mathcal{C}$  and  $\eta : Id \rightarrow T$  is a natural transformation.
- A *monad* is a triple  $\langle T, \eta, \mu \rangle$ , where  $\langle T, \eta \rangle$  is a pointed endofunctor,  $\eta$  is called the *unit* of the monad,  $\mu : T^2 \rightarrow T$  is a natural transformation, called the *multiplication* of the monad, such that the diagrams below commute:

$$\begin{array}{ccc}
 T & \xrightarrow{\eta T} & T^2 & \xleftarrow{T\eta} & T \\
 & \searrow id & \downarrow \mu & & \swarrow id \\
 & & T & & 
 \end{array}
 \qquad
 \begin{array}{ccc}
 T^2 & \xleftarrow{T\mu} & T^3 \\
 \mu \downarrow & & \downarrow \mu T \\
 T & \xleftarrow{\mu} & T^2
 \end{array}$$

**Definition 1.8** [Corecursion Monad] Let  $F : \mathcal{C} \rightarrow \mathcal{C}$  have final coalgebra  $(\Omega, \alpha_\Omega)$ . Let  $T_F^+ : \mathcal{C} \rightarrow \mathcal{C}$  be the functor defined by

$$\forall X. T_F^+ X = X + \Omega, \quad \forall f : X \rightarrow Y. T_F^+ f = [in_1 \circ f, in_2],$$

where  $in_1, in_2$  are the canonical sum injections.

The functor  $T_F^+$  can be endowed with a structure of monad  $\langle T_F^+, \eta, \mu \rangle$  by defining

$$\eta_X = in_1 : X \rightarrow X + \Omega, \quad \mu_X = [id, in_2] : (X + \Omega) + \Omega \rightarrow X + \Omega.$$

**Definition 1.9** [Variator, Free Monad]

- Let  $F : \mathcal{C} \rightarrow \mathcal{C}$  be such that the functor  $F_X^+$ , defined by  $\forall Z. F_X^+Z = X + FZ, \quad \forall f : Z_1 \rightarrow Z_2. F_X^+f = \text{id}_X + Ff$ , has initial algebra, for all  $X$ . We shall call such a functor *variator*.
- Let  $F$  be a variator. The *monad*  $\langle T_F, \eta_{T_F} \mu_{T_F} \rangle$  *freely generated* by  $F$  is defined as follows:
  - for all  $X$ ,  $T_F X$  is the carrier of the *initial  $F_X^+$ -algebra*, i.e. the *free  $F$ -algebra* on  $X$ ,  $\mu Z. X + FZ$ .
  - Let  $\phi_{T_F X} = [\gamma_{T_F X}, \beta_{T_F X}] : X + F(T_F X) \rightarrow T_F X$  be the isomorphism on the initial  $F_X^+$ -algebra. For any  $f : X \rightarrow Y$ ,  $T_F(f)$  is the unique morphism from  $(T_F X, \phi_{T_F X})$  into the  $F_X^+$ -algebra  $(T_F Y, \phi_{T_F Y} \circ (f + \text{id}_{F(T_F Y)}))$ .
  - $(\eta_{T_F})_X : X \rightarrow T_F X, \quad (\eta_{T_F})_X = \gamma_{T_F X}$ .
  - $(\mu_{T_F})_X : T_F^2 X \rightarrow T_F X, \quad (\mu_{T_F})_X = \widehat{\text{id}_{T_F X}}$ , where  $\widehat{\text{id}_{T_F X}}$  is the unique  $F$ -algebra morphism *extending*  $\text{id}_{T_F X}$ . Namely, for any morphism  $f : X \rightarrow T_F Y$ , there exists a *unique*  $\widehat{f} : T_F X \rightarrow T_F Y$  which *extends*  $f$ , i.e.  $\widehat{f} \circ (\eta_{T_F})_X = f$ , where  $\widehat{f}$  is defined as the *unique*  $F_X^+$ -algebra morphism from  $(T_F X, \phi_{T_F X})$  into the  $F_X^+$ -algebra  $(T_F Y, [\text{id}_{T_F Y}, \beta_{T_F Y}] \circ (f + \text{id}_{F(T_F Y)}))$ .

## 2 A Gallery of Circular Specifications

In this section we present a collection of circular specifications of functions into a coinductive datatype, which will be used as motivating examples in the rest of the paper. For simplicity all deal with streams. The reader can easily extend the list w.r.t. her/his favourite coinductive datatype. In the sequel, we will show how to capture  $\{\text{co,bi}\}$ algebraically the underlying coiterative schemata.

We recall that *streams* on a set  $A$  form a *final coalgebra* for the functor  $F_{S_A} : \text{Set} \rightarrow \text{Set}$  defined by  $F_{S_A} X = A \times X$  (with standard behaviour on morphisms):  $\text{Streams} \equiv (S_A, \langle \text{hd}, \text{tl} \rangle : S_A \rightarrow A \times S_A)$ , where  $\text{hd} : S_A \rightarrow A$  gives the first element of the stream (the *observation*)  $\text{tl} : S_A \rightarrow S_A$  returns the rest of the stream (the *next state*).

In particular, we will denote by  $S_{\mathbf{N}}$  the set of streams on natural numbers, and simply by  $F_S$  the corresponding functor  $F_{S_{\mathbf{N}}}$ .

**Example 2.1** The binary function  $\oplus : S_{\mathbf{N}} \times S_{\mathbf{N}} \rightarrow S_{\mathbf{N}}$  which gives the stream obtained by adding the input streams elementwise can be specified as follows:

$$\langle \text{hd}, \text{tl} \rangle (s \oplus t) = \langle \text{hd}(s) + \text{hd}(t), \text{tl}(s) \oplus \text{tl}(t) \rangle .$$

When applied to streams of real numbers corresponding to Taylor coefficients of two analytical functions  $f$  and  $g$ , the operation  $\oplus$  yields the Taylor series of the function  $f + g$ , [EP98,Rut00].

**Example 2.2** The function  $ms : S_{\mathbf{N}} \times S_{\mathbf{N}} \rightarrow S_{\mathbf{N}}$  specified as follows

$$\langle \text{hd}, \text{tl} \rangle (ms(s, t)) = \begin{cases} \langle \text{hd}(s), ms(\text{tl}(s), t) \rangle & \text{if } \text{hd}(s) < \text{hd}(t) \\ \langle \text{hd}(s), ms(\text{tl}(s), \text{tl}(t)) \rangle & \text{if } \text{hd}(s) = \text{hd}(t) \\ \langle \text{hd}(t), ms(s, \text{tl}(t)) \rangle & \text{if } \text{hd}(t) < \text{hd}(s) , \end{cases}$$

yields the familiar function *merge*, when applied to strictly monotone streams.

**Example 2.3** The function  $map_g : S_{\mathbf{N}} \rightarrow S_{\mathbf{N}}$ , for  $g : Nat \rightarrow Nat$ , which applies  $g$  to all the components of the input stream, can be specified by:

$$\langle hd, tl \rangle (map_g(s)) = \langle g(hd(s)), map_g(tl(s)) \rangle .$$

**Example 2.4** The function  $h_0 : S_{\mathbf{N}} \rightarrow S_{\mathbf{N}}$  which, given a stream  $s$ , yields the stream obtained by replacing the first component of  $s$  by 0, can be specified by:

$$\langle hd, tl \rangle (h_0(s)) = \langle 0, tl(s) \rangle .$$

**Example 2.5** The function  $acc : S_{\mathbf{N}} \rightarrow S_{\mathbf{N}}$  which gives the stream whose  $n$ -th component is the sum of the first  $n$  components of the input can be defined using a (simple) mutual recursive specification, where  $acc' : Nat \times S_{\mathbf{N}} \rightarrow S_{\mathbf{N}}$ :

$$\begin{cases} acc(s) = acc'(0, s) \\ \langle hd, tl \rangle \circ acc'(n, s') = \langle n + hd(s'), acc'(n + hd(s'), tl(s')) \rangle \end{cases}$$

**Example 2.6** The unary function *exch* which, given a stream  $s$ , exchanges its components pairwise, can be specified as follows:

$$(\text{id} \times \langle hd, tl \rangle)(\langle hd, tl \rangle (exch(s))) = \langle hd(tl(s)), \langle hd(s), exch(tl(tl(s))) \rangle \rangle .$$

**Example 2.7** The binary function  $\otimes : S_{\mathbf{N}} \times S_{\mathbf{N}} \rightarrow S_{\mathbf{N}}$  is specified by:

$$\langle hd, tl \rangle (s \otimes t) = \langle hd(s) \cdot hd(t), (s \otimes tl(t)) \oplus (tl(s) \otimes t) \rangle ,$$

where the operation  $\oplus$  is defined in Example 2.1. When applied to streams of real numbers corresponding to the Taylor coefficients of two analytical functions  $f$  and  $g$ ,  $\otimes$  computes the Taylor series for the functional product  $f \cdot g$  (see [Rut00]). The function  $\otimes$  has been extensively discussed in [Bar01].

**Example 2.8** A stream of *Hamming Numbers* contains all natural numbers, in increasing order, which do not have prime factors other than those from a given set. For primes 2 and 3, the corresponding stream of Hamming Numbers can be specified as the image of the function  $ham:1 \rightarrow S_{\mathbf{N}}$  (this example has been extensively discussed in [Bar01]):

$$\langle hd, tl \rangle (ham(*)) = \langle 1, ms(map_{\times 2}(ham(*)), map_{\times 3}(ham(*))) \rangle ,$$

$ms$  is as in Example 2.2, and  $\times 2, \times 3 : \mathbf{N} \rightarrow \mathbf{N}$  are the functions which double and triple the components of their arguments, and  $map_{-}$  is as in Example 2.3.

**Example 2.9** The function  $h : 1 \rightarrow S_{\mathbf{N}}$  defined by

$$\langle hd, tl \rangle (h(*)) = \langle 1, acc \circ h(*) \rangle ,$$

yields the stream  $s$ , where the first element  $s_1 = 1$ , and  $s_{n+1} = \sum_{i \leq n} s_i$ , for all  $n \geq 1$ .

**Example 2.10** The stream of *Fibonacci Numbers*,  $fib: 1 \rightarrow S_{\mathbf{N}}$ , can be defined using a mutual recursive specification as follows ( $\oplus$  is defined in Example 2.1.):

$$\begin{cases} \langle hd, tl \rangle (fib(*)) = \langle 1, fib'(*) \rangle \\ \langle hd, tl \rangle (fib'(*)) = \langle 1, fib(*) \oplus fib'(*) \rangle \end{cases}$$

**Example 2.11** An alternative definition for the stream of *Fibonacci Numbers* is given by  $fib : 1 \rightarrow S_{\mathbf{N}}$  defined by

$$\langle hd, tl \rangle (fib(*)) = \langle 1, p(in_1(fib(*))) \rangle ,$$

where  $p : S_{\mathbf{N}} + S_{\mathbf{N}} \rightarrow S_{\mathbf{N}}$  pairwise sums the elements of a right input stream, while, given a left input stream, it yields the stream whose head is the same as that of the input, and whose tail is the result of the application of  $p$  to the input stream viewed as a righthand argument, i.e.:

$$\begin{cases} p \circ in_1 = \langle hd, p \circ in_2 \rangle \\ p \circ in_2 = \langle hd + hd \circ tl, p \circ in_2 \circ tl \rangle . \end{cases}$$

### 3 Coiteration

The origin, and the success, of the *coalgebraic account* of coinductive functions lies in the fact that the condition satisfied by a final coalgebra morphism into a final coalgebra, i.e. the commutativity of the appropriate diagram, expresses directly a coiterative specification schema.

**Definition 3.1** [Coiteration Schema] Let  $(X, \alpha_X)$  be an  $F$ -coalgebra, and let  $(\Omega, \alpha_\Omega)$  be a final  $F$ -coalgebra. The *coiterative morphism* is the unique  $F$ -coalgebra morphism  $f : (X, \alpha_X) \rightarrow (\Omega, \alpha_\Omega)$ .

Examples 2.1–2.3 illustrate this point immediately.

The function  $\oplus$  is the *coiterative morphism* induced by the  $F_S$ -coalgebra  $(S_{\mathbf{N}} \times S_{\mathbf{N}}, \alpha_\oplus)$ , where  $F_S = Nat \times \_$  is the functor defined at the beginning of Section 2, and  $\alpha_\oplus(s, t) = \langle hd(s) + hd(t), \langle tl(s), tl(t) \rangle \rangle$ :

$$\begin{array}{ccc} S_{\mathbf{N}} \times S_{\mathbf{N}} & \xrightarrow{\alpha_\oplus} & F_S(S_{\mathbf{N}} \times S_{\mathbf{N}}) \\ \oplus \downarrow & & \downarrow F_S(\oplus) \\ S_{\mathbf{N}} & \xrightarrow{\langle hd, tl \rangle} & F_S(S_{\mathbf{N}}) \end{array}$$

the commutativity of the diagram corresponding exactly to the specification given in Example 2.1.

Similarly, the function  $ms$  is the coiterative morphism induced by the  $F_S$ -coalgebra  $(S_{\mathbf{N}} \times S_{\mathbf{N}}, \alpha_{ms})$ , where

$$\alpha_{ms}(s, t) = \begin{cases} \langle hd(s), \langle tl(s), t \rangle \rangle & \text{if } hd(s) < hd(t) \\ \langle hd(s), \langle tl(s), tl(t) \rangle \rangle & \text{if } hd(s) = hd(t) \\ \langle hd(t), \langle s, tl(t) \rangle \rangle & \text{if } hd(t) < hd(s) . \end{cases}$$

The function  $map_g$  is the coiterative morphism induced by the  $F_S$ -coalgebra  $(S_{\mathbf{N}}, \alpha_{map_g})$ , where  $\alpha_{map_g}(s) = \langle g(hd(s)), tl(s) \rangle$ . Also the function  $p$  used in Example 2.11 can be easily seen as a coiterative morphism.

However, these are the only examples in Section 2 which can be captured directly as standard coiteration schemata. Already the trivial specification of a *constant* function, or Example 2.4, which are not even circular, escape this format.

In order to overcome this limitation, the third author introduced  $T$ -coiteration schemata, [Len99]. In Section 4 we will see that by choosing suitable monads  $T$ , Examples 2.4 and 2.5 can be captured directly using these generalized schemata.

A more structured approach is necessary to deal with Example 2.7. That specification does not provide a standard coiterative definition because of the use of  $\oplus$  in the expression for the tail. The recursive call of  $\otimes$  is said to be *guarded* by an extra given *operation*  $\oplus$ . This operation naturally induces a structure of algebra on the final coalgebra of streams, and the equivalence induced by  $\otimes$  is a congruence w.r.t.  $\oplus$ . In Section 5, we will see how to recover  $\otimes$  as a  $T$ -coiterative morphism, by working in a suitable category of  $\lambda$ -bialgebras, where the algebra structure is induced by  $\oplus$ . The same treatment via  $\lambda$ -bialgebras is necessary to deal with Examples 2.8 and 2.11, albeit in a category of bialgebras satisfying a *generalized* pentagonal law. In order to handle Example 2.10 we need to combine two monads, in a bialgebraic setting, namely the one for mutual coiteration and the one to deal with guards. In fact, both the recursive calls of the functions  $fib$  and  $fib'$  in the definition of  $fib'$  appear guarded by the  $\oplus$ -operation. Finally, as we will see, the function  $h$  of Example 2.9 escapes a direct treatment in the setting of  $\lambda$ -bialgebras, because the guard  $acc$  is not a standard coiterative morphism. However, we will provide an “equivalent” guarded specification for  $h$ , admitting a treatment in the setting of  $\lambda$ -bialgebras.

## 4 $T$ -coiteration

In this section, we introduce the generalized coiteration schema of [Len99] (see also [LPW00]), called  $T$ -coiteration, for  $T$  pointed endofunctor. We show that it is very expressive and it subsumes many classical schemata, such as corecursion (i.e. the dual of primitive recursion), the schema dual to course-of-value iteration, and it can handle mutual coiteration. One can easily check that already the corecursion schema is *extensionally universal*, in the sense that it captures the graph of any definable function into a final coalgebra. However, as we have already pointed out, it is the *intensional expressivity* of the schema that we are after. The “rule of the game” is that of showing that a *given* coiterative specification does define *indeed* a (unique) total function, without any pre-processing of the specification.

Throughout this section we assume that the functor  $F : \mathcal{C} \rightarrow \mathcal{C}$  has final coalgebra  $(\Omega, \alpha_\Omega)$ .

**Definition 4.1** [ $\langle T, \eta \rangle$ -coiteration Schema] Let  $\langle T, \eta \rangle$  be a pointed endofunctor over  $\mathcal{C}$ . Then, for any  $F$ -coalgebra  $(TX, \alpha)$ , we can define the  $\langle T, \eta \rangle$ -coiterative morphism (or  $T$ -coiterative morphism for short)  $h : X \rightarrow \Omega$  as

$f \circ \eta_X$ , where  $f$  is the unique  $F$ -coalgebra morphism from  $(TX, \alpha)$  to the final  $F$ -coalgebra  $(\Omega, \alpha_\Omega)$ :

$$\begin{array}{ccccc} X & \xrightarrow{\eta_X} & TX & \xrightarrow{\alpha} & FTX \\ & \searrow h & \downarrow f & & \downarrow Ff \\ & & \Omega & \xrightarrow{\alpha_\Omega} & F(\Omega) \end{array}$$

A  $T$ -coiterative morphism is obtained by precomposing a standard coiterative morphism with a suitable morphism. As is often the case in dealing with inductive issues, also here we have that a more “complex” function, obtained from the intended one using suitable operations, satisfies a “simpler” schema.

In Definition 4.1, we assumed  $T$  to be only a pointed endofunctor. But in many interesting examples of  $T$ -coiteration, the  $T$  used is actually a monad.

The function  $acc$  of Example 2.5 can be easily recovered by  $T$ -coiteration by taking as pointed endofunctor the functor  $F_S$  together with  $\eta_X : X \rightarrow Nat \times X$  to be  $\eta_X(x) = \langle 0, x \rangle$ . Then  $acc = acc' \circ \eta_{S_{\mathbf{N}}}$ , where  $acc'$  is the standard coiterative morphism induced by the  $F_S$ -coalgebra  $(Nat \times S_{\mathbf{N}}, \alpha_{acc'})$ , where  $\alpha_{acc'}(n, s) = \langle n + hd(s), \langle n + hd(s), tl(s) \rangle \rangle$ .

Constant functions  $K_{\bar{s}}$  into streams  $S_A$  can be immediately shown to be captured by  $T$ -coiteration, by taking  $T$  to be the pointed endofunctor constantly equal to  $S_A$  together with the natural transformation  $\eta$  defined by  $\eta_X(x) = \bar{s}$ . Then  $K_{\bar{s}} = id_{S_A} \circ \eta_{S_A}$  (the identity being a standard coiterative morphism).

#### 4.1 The Corecursion Schema

The *corecursion schema*, i.e. the dual of primitive recursion (see e.g. [UV99]), can be recovered by considering the corecursion monad of Definition 1.8:

**Definition 4.2** [Corecursion Schema] The *corecursion schema* is obtained by considering the pointed endofunctor  $\langle T_F^+, in_1 \rangle$  underlying the corecursion monad, and by taking  $F$ -coalgebras of the shape  $(X + \Omega, [\alpha_1, F(in_2) \circ \alpha_\Omega])$ , where  $\alpha_1 : X \rightarrow F(X + \Omega)$ :

$$\begin{array}{ccc} X & \xrightarrow{in_1} & X + \Omega \xrightarrow{[\alpha_1, F(in_2) \circ \alpha_\Omega]} F(X + \Omega) \\ & \searrow h_0 & \downarrow f \qquad \qquad \qquad \downarrow Ff \\ & & \Omega \xrightarrow{\alpha_\Omega} F\Omega \end{array}$$

Notice in particular that  $f \circ in_2 = id_\Omega$ .

The essence of this schema is that we can choose between the possibilities offered by the two branches of the disjoint sum in the  $F$ -coalgebra, where the morphism on the second branch essentially acts as the identity. E.g. the function  $h_0$  of Example 2.4 can be viewed immediately as an instance of corecursion, by taking  $F$  to be  $F_S$ , and  $\alpha_1 : S_{\mathbf{N}} \rightarrow S_{\mathbf{N}}$  to be defined by  $\alpha_1(s) = \langle 0, in_2(tl(s)) \rangle$ .

The corecursion schema is trivially *universal* in the following sense:

**Proposition 4.3 (Universality of Corecursion)** *Let  $f : X \rightarrow \Omega$  be a morphism in  $\mathcal{C}$ . Then  $f$  can be viewed as the  $\langle T_F^+, \text{in}_1 \rangle$ -coiterative morphism induced by the coalgebra  $(X + \Omega, [\alpha_1, \alpha_2])$ , where:  $\alpha_1 = F(\text{in}_2) \circ \alpha_\Omega \circ f$  and  $\alpha_2 = F(\text{in}_2) \circ \alpha_\Omega$ .*

Of course the universality of Proposition 4.3 is not particularly useful, since we are assuming that the morphism is already definable. The following expressivity results have a much stronger impact.

#### 4.2 The Schema Dual to Course-of-value Iteration

The dual to the course-of-value iteration schema (see e.g. [UV99]) can be viewed as an instance of  $T$ -coiteration by considering the pointed endofunctor underlying the free monad  $T_F$  of Definition 1.9, as follows:

**Definition 4.4** [Dual to Course-of-value Iteration] The *schema dual to course-of-value iteration* is obtained by considering the pointed endofunctor  $T_F$  underlying the free monad, and by taking  $F$ -coalgebras of the shape  $(T_F X, [\alpha_1, \text{id}_{F(T_F X)}] \circ \phi_{T_F X}^{-1})$ , where  $\alpha_1 : X \rightarrow F(T_F X)$ :

$$\begin{array}{ccc} X & \xrightarrow{\gamma_{T_F X}} & T_F X & \xrightarrow{[\alpha_1, \text{id}_{F(T_F X)}] \circ \phi_{T_F X}^{-1}} & F(T_F X) \\ & & \downarrow f & & \downarrow Ff \\ & & \Omega & \xrightarrow{\alpha_\Omega} & F(\Omega) \end{array}$$

The schema above is only apparently complex, the basic idea in the case of streams being simple. Course-of-value iteration allows to define the value of a function on a given object using its value on subcomponents of arbitrary depth, and not only on the immediate ones. In the dual schema, in specifying the behaviour of the output we are allowed to mention stages arbitrarily further on in the future of the output, instead of being forced to use only the current stage.

A simple example of a function on streams which is defined by such a schema is the function *exch* of Example 2.6. This is obtained by taking  $F$  to be  $F_S$ ,  $X$  to be  $S_{\mathbf{N}}$ , and  $\alpha_1 : S_{\mathbf{N}} \rightarrow \text{Nat} \times T_{F_S}(S_{\mathbf{N}})$  to be defined by  $\alpha_1(s) = \langle \text{hd}(tl(s)), \beta_{T_{F_S} S_{\mathbf{N}}}(\text{hd}(s), \gamma_{T_{F_S} S_{\mathbf{N}}}(tl(tl(s)))) \rangle$ , where  $\beta_{T_{F_S} S_{\mathbf{N}}}, \gamma_{T_{F_S} S_{\mathbf{N}}}$  are as in Definition 1.9.

#### 4.3 Definitions by Mutual Recursion

Here we show how to capture by  $T$ -coiteration elementary mutual specifications. Complex ones will be discussed in Section 5.5.

Consider the following specification of the mutually defined functions  $h_1, \dots, h_k$ , where  $h_i : X_i \rightarrow \Omega$ , and  $\alpha_i : X_i \rightarrow F(\coprod_{i=1}^k X_i)$ , for  $i = 1, \dots, k$ :

$$\left\{ \begin{array}{l} \alpha_\Omega \circ h_1 = F([h_1, \dots, h_k]) \circ \alpha_1 \\ \dots \\ \alpha_\Omega \circ h_k = F([h_1, \dots, h_k]) \circ \alpha_k \end{array} \right.$$

The function  $[h_i]_{i=1}^k : \coprod_{i=1}^k X_i \rightarrow \Omega$  can be captured by standard coiteration by precomposing with the canonical injection of  $X_i$  into  $\coprod_{i=1}^k X_i$ .

Notice that the above schema subsumes immediately the corecursion schema. It subsumes also the dual of course-of-value iteration, since one can check that this can be equivalently expressed by considering the pointed endofunctor  $\langle - + FT_F(-), in_1 \rangle$  together with coalgebras of the shape  $(X + F(T_F X), [\alpha_1, F(\phi_{T_F X}^{-1})])$ .

## 5 Working in a Bialgebraic Setting: i.e. Explaining Guards as Algebraic Operations

In this section, we show how to capture by  $T$ -coiteration a relevant class of circularly defined functions into final coalgebras, where recursive calls are guarded by extra operations on the final coalgebra. The specifications in Examples 2.7–2.11 are of this kind. The crucial move is to rework the  $T$ -coiteration paradigm in a more structured ambient category: that of suitable bialgebras, where the algebra structure is induced by the guard operation. In particular, we introduce a *uniform* criterion for a guarded specification to actually define a function, in terms of conditions on the guard. We discuss briefly a possible *effective* implementation of such a criterion for streams. In this section, we consider only specifications involving a single function. Mutual guarded specifications will be considered in Section 5.5. Throughout this section we assume  $F$  and  $G$  to be endofunctors in the same category  $\mathcal{C}$ , and moreover  $G$  to be good.

### 5.1 A Class of Guarded Specifications

The class of guarded specifications that we consider is the following:

**Definition 5.1** [Guarded Specification] Let  $(\Omega, \alpha_\Omega)$  be a final  $F$ -coalgebra. A *guarded specification* for a morphism  $h : X \rightarrow \Omega$  is of the form:

$$\alpha_\Omega \circ h = F(g \circ G(h)) \circ \delta , \tag{1}$$

where  $\delta : X \rightarrow F(GX)$  and  $g : G(\Omega) \rightarrow \Omega$  are given,  $g$  is the *guard*.

Notice that we do not know a priori whether a guarded specification defines a function at all, and in the case it defines a function, whether this is unique. The following are examples of “invalid” specifications.

**Example 5.2** Let  $g : S_{\mathbf{N}} \rightarrow S_{\mathbf{N}}$  be the function which adds the elements of a given stream pairwise, i.e.  $\langle hd, tl \rangle \circ g(s) = \langle hd(s) + hd(tl(s)), g(tl(tl(s))) \rangle$ . This is a standard coiterative morphism. Notice that  $g$  “consumes” two observations in order to produce the first element of the output stream. However, there is no function  $h : 1 \rightarrow S_{\mathbf{N}}$  satisfying the following guarded specification:

$$\langle hd, tl \rangle (h(*)) = \langle 1, g(h(*)) \rangle .$$

The function  $h$  produces immediately the first component  $a_1$  of the output stream, i.e. 1. But, by definition of  $g$ , the second component  $a_2$  of the output should satisfy the specification  $a_2 = 1 + a_2$ , which has no solution in  $Nat$ .

$$\begin{array}{ccccc}
 & & X & & \\
 & & \downarrow \gamma_{T_G X} & & \\
 G(T_G X) & \xrightarrow{\beta_{T_G X}} & T_G X & \xrightarrow{\alpha_{T_G X}} & F(T_G X) \\
 \downarrow Gf & & \downarrow f & & \downarrow Ff \\
 G(\Omega) & \xrightarrow{g} & \Omega & \xrightarrow{\alpha_\Omega} & F\Omega
 \end{array}
 \quad (1) \qquad (2)$$

Fig. 1. Bialgebraic interpretation of a guarded specification.

**Example 5.3** The following is a guarded specification admitting infinitely many distinct solutions:  $\langle hd, tl \rangle(h(*)) = \langle 0, g(h(*)) \rangle$ . E.g. the stream  $\langle 0, n, 0, 0, \dots \rangle$  is a solution for any  $n$ .

In view of the above examples, a sufficient condition to guarantee that a guarded specification actually defines a (unique) function has to be a form of *productivity* (or *effectiveness*) of the guard. I.e. a guard on streams should consume at most the first  $n$  observations in order to produce the  $n$ -th output observation. In what follows, we formalize this by giving conditions on the guard  $g$  for the specified function to be  $T$ -coiterative in a *bialgebraic setting*.

The general pattern that we utilize can be roughly described as follows.

First of all, notice that  $g$  determines a structure of  $G$ -algebra on  $\Omega$ . That is  $(\Omega, g, \alpha_\Omega)$  is a  $\langle G, F \rangle$ -bialgebra. A form of productivity condition on  $g$  can thus be expressed by requiring that the  $\langle G, F \rangle$ -bialgebra  $(\Omega, g, \alpha_\Omega)$  is a  $\lambda$ -bialgebra for a (special kind of) generalized distributive law  $\lambda$ . If this is the case, then, by Proposition 1.6,  $(\Omega, g, \alpha_\Omega)$  is necessarily a *final  $\lambda$ -bialgebra*. Moreover, let us assume that the free  $G$ -algebra on  $X$ ,  $T_G X = \mu Z.X + G(Z)$ , can also be endowed with a structure of  $\lambda$ -bialgebra  $(T_G X, \beta_{T_G X}, \alpha_{T_G X})$ , where the algebra part is given by  $\beta_{T_G X} : G(T_G X) \rightarrow T_G X$  (see Definition 1.9), while the coalgebra part is induced by  $\lambda$  and  $\delta$  in a suitable way. Then one can show that the  $T_G$ -coiterative morphism  $f \circ \gamma_{T_G X}$ , where  $f$  is the unique  $\lambda$ -bialgebra morphism from  $(T_G X, \beta_{T_G X}, \alpha_{T_G X})$  to  $(\Omega, g, \alpha_\Omega)$  (see Fig. 1), is the unique solution of the guarded specification induced by  $g$  and  $\delta$  (see Theorem 5.4 below).

Putting formally the above we have:

**Theorem 5.4** *Let  $\delta : X \rightarrow F(GX)$ , and let  $g : G\Omega \rightarrow \Omega$ . If*

(i) *there exists a generalized distributive law  $\lambda : GH \rightarrow FG$  for which  $(\Omega, g, \alpha_\Omega)$  is a  $\lambda$ -bialgebra,*

(ii) *the free  $G$ -algebra  $T_G X$  can be endowed with a structure of  $\lambda$ -bialgebra  $(T_G X, \beta_{T_G X}, \alpha_{T_G X})$ , where  $\alpha_{T_G X} : T_G X \rightarrow F(T_G X)$  is such that  $\alpha_{T_G X} \circ \gamma_{T_G X} = F(\beta_{T_G X} \circ G(\gamma_{T_G X})) \circ \delta$ ,*

*then the morphism  $f \circ \gamma_{T_G X}$ , where  $f$  is the unique bialgebra morphism from  $(T_G X, \beta_{T_G X}, \alpha_{T_G X})$  to  $(\Omega, g, \alpha_\Omega)$ , is the unique solution of the guarded specification induced by  $g$  and  $\delta$ .*

**Proof.** By first exploiting that  $f$  is an  $F$ -coalgebra homomorphism (i.e. by

commutativity of diagram (2) in Figure 1), and then that  $f$  is a  $G$ -algebra homomorphism (i.e. by the commutativity of diagram (1) in Figure 1), using hypothesis (ii) on  $\alpha_{T_G}$ , we have:

$\alpha_\Omega \circ f \circ \gamma_{T_G X} \stackrel{(2)}{=} F(f) \circ \alpha_{T_G X} \circ \gamma_{T_G X} \stackrel{(\text{hyp. (ii)})}{=} F(f) \circ F(\beta_{T_G X} \circ G(\gamma_{T_G X})) \circ \delta \stackrel{(1)}{=} F(g \circ G(f \circ \gamma_{T_G X})) \circ \delta$ , i.e.  $f \circ \gamma_{T_G X}$  satisfies the guarded specification. Moreover, this is the unique solution, since, if  $h : X \rightarrow \Omega$  satisfies the guarded specification, then  $h = f \circ \gamma_{T_G X}$ , where  $f$  is the unique final morphism above. Namely, let  $f : T_G X \rightarrow \Omega$  be the unique  $X + G(\ )$ -algebra morphism from  $(T_G X, \phi_{T_G X})$  to  $(\Omega, [h, g])$ . Then  $h = f \circ \gamma_{T_G X}$ . Moreover, using hypothesis (ii) and the injectivity of  $\gamma_{T_G X}$ , one can show that  $f$  is the unique final morphism of Figure 1.  $\square$

A strong form of productivity of the guard  $g$  is obtained when  $(\Omega, g, \alpha_\Omega)$  is a bialgebra for a  $\lambda_I$ -distributive law with  $I \subseteq \{0, 1\}$ .

**Definition 5.5** [Strongly Productive Operation] Let  $(\Omega, \alpha_\Omega)$  be a final  $F$ -coalgebra. An operation  $g : G(\Omega) \rightarrow \Omega$  is *strongly productive* if  $(\Omega, g, \alpha_\Omega)$  is a  $\lambda_I$ -bialgebra for a distributive law  $\lambda_I$  with  $I \subseteq \{0, 1\}$ .

If the distributive law is a  $\lambda_I$ -distributive law for  $I \subseteq \{0, 1\}$ , then hypothesis (ii) of Theorem 5.4 is automatically verified:

**Lemma 5.6** Let  $\delta : X \rightarrow F(GX)$ , and let  $\lambda_I : G(\prod_{i \in I} F^i) \rightarrow FG$  be a distributive law for  $I \subseteq \{0, 1\}$ . Then the free  $G$ -algebra  $T_G X$  can be endowed with a structure of  $\lambda_I$ -bialgebra  $(T_G X, \beta_{T_G X}, \alpha_{T_G X})$ , where  $\alpha_{T_G X} : T_G X \rightarrow F(T_G X)$  is defined by “induction on terms” of the free algebra by:

- $\alpha_{T_G X} \circ \gamma_{T_G X} = F(\beta_{T_G X} \circ G(\gamma_{T_G X})) \circ \delta$
- $\alpha_{T_G X} \circ \beta_{T_G X} = F(\beta_{T_G X}) \circ (\lambda_I)_{T_G X} \circ G(\langle (\alpha_{T_G X})_i \rangle_{i \in I})$ .

An immediate consequence of Theorem 5.4 and Lemma 5.6 is:

**Theorem 5.7** Let  $g : G\Omega \rightarrow \Omega$ , and  $\delta : X \rightarrow F(GX)$ . If  $g$  is strongly productive, then the morphism  $f \circ \gamma_{T_G X}$ , where  $f$  is the unique bialgebra morphism from  $(T_G X, \beta_{T_G X}, \alpha_{T_G X})$  to  $(\Omega, g, \alpha_\Omega)$ , satisfies the guarded specification induced by  $g$  and  $\delta$ .

Theorem 5.7 gives us a criterion for validating guarded specifications: “Given a guarded specification, in order to prove that it defines a ( $T$ -coiterative) function, we only need to check that the guard is strongly productive”.

The interest of this criterion is that it can be easily made effective in many cases. To illustrate this we focus on streams, and give a simple sufficient *syntactic* condition on the guard so that it is uniformly productive. This method could be strengthened and probably generalized to deal uniformly with final coalgebras of *polynomial* functors.

**Proposition 5.8** Let  $g : S_A^n \rightarrow S_A$ . If  $g$  is defined by

$$\langle hd, tl \rangle \circ g = \langle \epsilon_0 \circ hd^n, g \circ (\epsilon_1 \times \dots \times \epsilon_n) \rangle$$

where  $\epsilon_0 : A^n \rightarrow A$ , and, for all  $i = 1, \dots, n$ ,  $\epsilon_i : S_A \rightarrow S_A$  is either  $tl$  or  $id_{S_A}$ , then  $g$  is a standard coiterative uniformly productive operation.

### 5.2 Representing $\otimes$ by $T$ -coiteration.

Now we apply Theorem 5.7 to Example 2.7. By Theorem 5.7, we only have to check that  $\oplus$  is strongly productive, i.e. we have to find a distributive law  $\lambda_{\oplus}$  such that  $(S_{\mathbf{N}}, \oplus, \langle hd, tl \rangle)$  is a  $\lambda_{\oplus}$ -bialgebra:

**Lemma 5.9** *The  $\langle G, F_S \rangle$ -bialgebra  $(S_{\mathbf{N}}, \oplus, \langle hd, tl \rangle)$ , where  $G = Id \times Id$ , is a  $\lambda_{\oplus}$ -bialgebra for  $\lambda_{\oplus} : GF_S \rightarrow F_S G$  defined by:*

$$(\lambda_{\oplus})_X(\langle n, x \rangle, \langle n', x' \rangle) = \langle n + n', \langle x, x' \rangle \rangle .$$

By Lemma 5.6, we can endow  $T_G(S_{\mathbf{N}}^2) = \mu Z.S_{\mathbf{N}}^2 + Z^2$  with a structure of  $\lambda_{\oplus}$ -bialgebra  $(T_G(S_{\mathbf{N}}^2), \beta_{T_G(S_{\mathbf{N}}^2)}, \alpha_{T_G(S_{\mathbf{N}}^2)})$ , where  $\alpha_{T_G(S_{\mathbf{N}}^2)}$  is defined as follows:

- for  $s, t \in S_{\mathbf{N}}$ ,  
 $\alpha_{T_G(S_{\mathbf{N}}^2)}(\gamma_{T_G(S_{\mathbf{N}}^2)}(s, t)) = \langle hd(s) \cdot hd(t), \beta_{T_G(S_{\mathbf{N}}^2)}(\gamma_{T_G(S_{\mathbf{N}}^2)}(s, tl(t)), \gamma_{T_G(S_{\mathbf{N}}^2)}(tl(s), t)) \rangle$ .
- for  $\sigma, \tau \in T_G(S_{\mathbf{N}}^2)$ ,  
 $\alpha_{T_G(S_{\mathbf{N}}^2)}(\beta_{T_G(S_{\mathbf{N}}^2)}(\sigma, \tau)) =$   
 $\langle \pi_1(\alpha_{T_G(S_{\mathbf{N}}^2)}(\sigma)) + \pi_1(\alpha_{T_G(S_{\mathbf{N}}^2)}(\tau)), \beta_{T_G(S_{\mathbf{N}}^2)}(\pi_2(\alpha_{T_G(S_{\mathbf{N}}^2)}(\sigma)), \pi_2(\alpha_{T_G(S_{\mathbf{N}}^2)}(\tau))) \rangle$ .

Finally, by Theorem 5.7, we have:

**Proposition 5.10** *The generalized coiterative morphism  $f \circ \gamma_{T_G(S_{\mathbf{N}}^2)}$ , where  $f$  is the unique morphism from the  $\lambda_{\oplus}$ -bialgebra  $(T_G(S_{\mathbf{N}}^2), \beta_{T_G(S_{\mathbf{N}}^2)}, \alpha_{T_G(S_{\mathbf{N}}^2)})$  into the final  $\lambda_{\oplus}$ -bialgebra  $(S_{\mathbf{N}}, \oplus, \langle hd, tl \rangle)$ , satisfies the specification of Example 2.7.*

### 5.3 Representing the Stream of Hamming Numbers by $T$ -coiteration.

In the specification of Example 2.8, the recursive calls of the function *ham* are guarded by the binary operation on streams  $m : S_{\mathbf{N}} \times S_{\mathbf{N}} \rightarrow S_{\mathbf{N}}$ , defined by:

$$m = ms \circ (\text{map}_{\times 2} \times \text{map}_{\times 3}) .$$

Notice in particular that  $m$ , being defined as composition of coiterative functions is itself coiterative. Now, in order to define the function *ham* as a  $T_G$ -coiterative morphism, we apply Theorem 5.7 for  $G = Id \times Id$ . In this case we need a truly generalized distributive law  $\lambda_m : G(Id \times F_S) \rightarrow F_S G$ , since  $ms$  (and hence  $m$ ) does not necessarily consume the first component of both the input streams.

**Lemma 5.11** *The  $\langle G, F_S \rangle$ -bialgebra  $(S_{\mathbf{N}}, m, \langle hd, tl \rangle)$  is a  $\lambda_m$ -bialgebra for  $G = Id \times Id$  and  $\lambda_m : G(Id \times F_S) \rightarrow F_S G$  the generalized distributive law defined by:*

$$(\lambda_m)_X(\langle x, \langle n, x' \rangle \rangle, \langle y, \langle n', y' \rangle \rangle) = \begin{cases} \langle 2n, \langle x', y \rangle \rangle & \text{if } 2n < 3n' \\ \langle 2n, \langle x', y' \rangle \rangle & \text{if } 2n = 3n' \\ \langle 3n', \langle x, y' \rangle \rangle & \text{if } 2n > 3n' . \end{cases}$$

By Lemma 5.6,  $(T_G 1, \beta_{T_G 1}, \alpha_{T_G 1})$  is a  $\lambda_m$ -bialgebra, where the  $F_S$ -coalgebra morphism  $\alpha_{T_G 1} : T_G(1) \rightarrow \text{Nat} \times T_G(1)$  is defined as follows:

- $\alpha_{T_G 1}(\gamma_{T_G 1}(*)) = \langle 1, \beta_{T_G 1}(\gamma_{T_G 1}(*), \gamma_{T_G 1}(*)) \rangle$ .
- for  $\sigma, \tau \in T_G 1$ ,  $\alpha_{T_G 1}(\beta_{T_G 1}(\sigma, \tau)) =$

$$\left\{ \begin{array}{ll} \langle 2\pi_1(\alpha_{TG1}(\sigma)), \beta_{TG1}(\pi_2(\alpha_{TG1}(\sigma)), \tau) \rangle & \text{if } 2\pi_1(\alpha_{TG1}(\sigma)) < 3\pi_1(\alpha_{TG1}(\tau)) \\ \langle 2\pi_1(\alpha_{TG1}(\sigma)), \beta_{TG1}(\pi_2(\alpha_{TG1}(\sigma)), \pi_2(\alpha_{TG1}(\tau))) \rangle & \text{if } 2\pi_1(\alpha_{TG1}(\sigma)) = 3\pi_1(\alpha_{TG1}(\tau)) \\ \langle 3\pi_1(\alpha_{TG1}(\tau)), \beta_{TG1}(\sigma, \pi_2(\alpha_{TG1}(\tau))) \rangle & \text{if } 2\pi_1(\alpha_{TG1}(\sigma)) > 3\pi_1(\alpha_{TG1}(\tau)) \end{array} \right. .$$

#### 5.4 Beyond Strongly Productive Guards

Theorem 5.7 allows us to deal with many interesting guarded specifications. However, there are many valid specifications whose guards are not strongly productive. Some of them do not even induce a generalized distributive law. Namely, if the guard is not standard coiterative, then it cannot satisfy any (generalized) distributive law. This is the case of Example 2.9. However, the guard  $acc$  of Example 2.9 is  $T$ -coiterative, as we have seen in Section 4. We can deal with it in our setting, by considering an “equivalent” specification, obtained by slightly manipulating the original one in such a way that the new guard is the standard coiterative morphism appearing in the  $T$ -coiteration schema of  $acc$ .

The following proposition gives a general technique for transforming a guarded specification whose guard is a  $T$ -coiterative morphism into an equivalent specification with a standard coiterative guard:

**Proposition 5.12** *Let  $\delta : X \rightarrow FGX$ , let  $g : G\Omega \rightarrow \Omega$  be a  $\langle T, \eta \rangle$ -coiterative morphism, i.e.  $g = g' \circ \eta_{G\Omega}$ , for some standard coiterative morphism  $g' : TG\Omega \rightarrow \Omega$ . Then the guarded specification induced by  $\delta$  and  $g$  is equivalent (i.e. it has the same solutions) to that induced by  $\delta'$  and  $g'$ , where  $\delta' : X \rightarrow FTGX$ ,  $\delta' \triangleq F(\eta_{GX}) \circ \delta$ .*

**Proof.** (Sketch)  $F(g \circ Gh) \circ \delta = F(g' \circ \eta_{G\Omega} \circ Gh) \circ \delta = F(g' \circ TGh \circ \eta_{GX}) \circ \delta = F(g' \circ TGh) \circ \delta'$ , using naturality of  $\eta$ .  $\square$

Hence, by Proposition 5.12, the guarded specification of Example 2.9 is equivalent to the specification  $\langle hd, tl \rangle \circ h = \langle id_{\mathbf{N}}, acc' \circ (id_{\mathbf{N}} \times h) \rangle \circ \delta'$ , where  $\delta' : 1 \rightarrow \mathbf{N} \times (\mathbf{N} \times 1)$  is defined by  $\langle \pi_1 \circ \delta, \eta_1 \rangle$ , and  $\delta : 1 \rightarrow \mathbf{N} \times 1$ ,  $\delta(*) = \langle 1, * \rangle$ . Now  $acc'$  is a standard coiterative morphism and one can easily check that there is a distributive law à la Plotkin-Turi accounting for it.

Another significant case, which escapes the criterion of strong productivity, is that of Example 2.11. There the guard is standard coiterative, but one can easily check that it is not strongly productive. However, there exists a distributive law  $\lambda_I$ , for  $I = \{1, 2\}$ , such that  $(S_{\mathbf{N}}, p, \langle hd, tl \rangle)$  is a  $\lambda_I$ -bialgebra. One can check that also hypothesis (ii) of Theorem 5.4 holds, and hence the specification of Example 2.11 has a unique solution. Examples like the above are precisely those which motivated the full generality in Definition 1.4.

#### 5.5 A More Complex Example

In order to capture the function  $fib$  of Example 2.10 by  $T$ -coiteration, we compose the monad corresponding to mutual recursion of Section 4.3 with the free monad used to deal with guarded specifications of Section 5.1 as follows:

**Proposition 5.13** *The function fib of Example 2.10 above is the  $T$ -coiterative function  $f \circ \gamma_{T_G(1+1)} \circ in_1$ , making the following diagram commute:*

$$\begin{array}{ccccc}
 & & 1 & & \\
 & & \downarrow \gamma_{T_G(1+1)} \circ in_1 & & \\
 (T_G(1+1))^2 & \xrightarrow{\beta_{T_G(1+1)}} & T_G(1+1) & \xrightarrow{\alpha_{T_G(1+1)}} & Nat \times T_G(1+1) \\
 \downarrow f \times f & & \downarrow f & & \downarrow id_{\mathbf{N}} \times f \\
 S_{\mathbf{N}}^2 & \xrightarrow{\oplus} & S_{\mathbf{N}} & \xrightarrow{\langle hd, tl \rangle} & Nat \times S_{\mathbf{N}}
 \end{array}$$

where  $G = Id \times Id$ ,  $f$  is the final  $\lambda_{\oplus}$ -bialgebra morphism from the  $\lambda_{\oplus}$ -bialgebra  $(T_G(1+1), \beta_{T_G(1+1)}, \alpha_{T_G(1+1)})$  into the final  $\lambda_{\oplus}$ -bialgebra  $(S_{\mathbf{N}}, \oplus, \langle hd, tl \rangle)$ .  $\alpha_{T_G(1+1)} : T_G(1+1) \rightarrow Nat \times T_G(1+1)$  is defined as follows:

- $\alpha_{T_G(1+1)}(\gamma_{T_G(1+1)}(in_1(*))) = \langle 1, \gamma_{T_G(1+1)}(in_2(*)) \rangle$
- $\alpha_{T_G(1+1)}(\gamma_{T_G(1+1)}(in_2(*))) = \langle 1, \beta_{T_G(1+1)}(in_1(*), in_2(*)) \rangle$
- for  $\sigma, \tau \in T_G(1+1)$ ,  $\alpha_{T_G(1+1)}(\beta_{T_G(1+1)}(\sigma, \tau)) = \langle \pi_1 \circ \alpha_{T_G(1+1)}(\sigma) + \pi_1 \circ \alpha_{T_G(1+1)}(\tau), \beta_{T_G(1+1)}(\pi_2 \circ \alpha_{T_G(1+1)}(\sigma), \pi_2 \circ \alpha_{T_G(1+1)}(\tau)) \rangle$

## 6 Final Remarks and Directions for Future Work

In this paper we have presented a bialgebraic semantics for a general schema for (mutually) defining recursive functions into coinductive types. Here is a list of problems which remain to be addressed.

- As we have seen in Subsection 5.4, the strong productivity condition is only a sufficient condition for a guarded specification to have a unique solution. A more general effective criterion involving both the parameter  $\delta$  and the guard  $g$  in the specification is called for.
- The example of Subsection 5.5 can be viewed as an instance of a general guarded schema obtained by combining the monad for mutual recursion and the free monad for guarded specifications. This should provide a compositional coalgebraic semantics to a language for processing coinductive datatypes whose basic building block is *standard coiteration*. This pattern could be extended to a fullfledged “total” language in the style of Barbier [Bar02]. For instance we could discuss the possibility of including the very mutually defined functions as guards. As we pointed out earlier, this has an important application in interactive proof development environments based on Type Theory. Currently the coiterative patterns accepted there are less expressive than the one we introduce in Section 5.5, since only definitions where recursive calls are *guarded by constructors* are accepted.
- It would be interesting to contrast the present approach to coiteration schemata to the one underpinning languages such as *Charity* of [CS95].
- We mentioned in the Introduction that *bisimulations up-to* cannot be captured directly in terms of standard *coalgebraic bisimulations*. But, as shown in [Len99, Bar01] a suitable categorical generalization can still be provided i.e. *bisimulation up-to T*. These exploit an underlying structure of  $T$ -algebra, which was achieved in [Len99] by assuming that  $T$  is a monad, and in [Bar01]

by working on bialgebras at the outset. More investigation is necessary here to achieve a complete picture.

## References

- [Acz88] P.Aczel. *Non-well-founded sets*, CSLI Lecture Notes **14**, Stanford 1988.
- [Acz93] P.Aczel. *Final Universes of Processes*, *MFPS'93*, Brookes et al. eds., LNCS **802**, 1993.
- [Bar02] B.Barbier. Solving stream equation systems, *JFLA*, 2002.
- [Bar01] F.Bartels. Generalised coinduction, *CMCS'01*, A.Corradini, M.Lenisa and U.Montanari eds., ENTCS **44**, 2001.
- [BM96] J.Barwise, L.Moss. *Vicious circles: On the mathematics of non-wellfounded phenomena*, CSLI Publications, Stanford, 1996.
- [CS95] R.Cockett, D.Spencer. Strong categorical datatypes II: A term logic for categorical programming, *TCS* **139**, 1995, 69–113.
- [Coq94] T.Coquand. Infinite Objects in Type Theory, *TYPES-93*, LNCS **806**.
- [CHM01] A.Corradini, R.Heckel, U.Montanari. Compositional SOS and beyond: A coalgebraic view of open systems, to appear in *TCS*.
- [EP98] M.Escardo, D.Pavlovic, Calculus in Coinductive Form, *LICS'98*, IEEE, Computer Science Press, 1998.
- [Gim94] E.Giménez. Codifying guarded definitions with recursive schemata, *TYPES*, P.Dybjer et al. eds., LNCS **996**, 1994, 39–59.
- [HLMP98] F.Honsell, M.Lenisa, U.Montanari, M.Pistore. Final Semantics for the  $\pi$ -calculus, *PROCOMET'98*, D. Gries et al. eds, Chapman & Hall, 1998.
- [JR96] B.Jacobs, J.Rutten. A tutorial on (co)algebras and (co)induction, *Bulletin of the EATCS* **62**, 1996, 222–259.
- [Len96] M.Lenisa. Final Semantics for a Higher Order Concurrent Language, *CAAP'96*, H.Kirchner et. al. eds., LNCS **1059**, 1996, 102–118.
- [Len99] M.Lenisa. From Set-theoretic Coinduction to Coalgebraic Coinduction: some results, some problems, *CMCS'99*, B.Jacobs and J.Rutten eds., ENTCS **19**, 1999.
- [LPW00] M.Lenisa, J.Power, H.Watanabe. Distributivity for endofunctors, pointed and co-pointed endofunctors, monads and comonads, *CMCS'00*, B. Jacobs and J. Rutten eds., ENTCS **33**, 2000.
- [Pav98] D.Pavlovic. Guarded induction on final coalgebras, *CMCS'98*, B.Jacobs et al eds., ENTCS **11**, 1998.
- [Rut00] J.J.M.M.Rutten. Behavioural differential equations: a coinductive calculus of streams, automata, and power series, *TR SEN-R0023*, CWI.
- [RT94] J.J.M.M.Rutten, D.Turi. *REX Conference Proceedings*, J.de Bakker et al. eds., LNCS **803**, 1994, 530–582.
- [San98] D.Sangiorgi. On the bisimulation proof method, *Math. Struct. In Comp. Science* **98**(8), 1998, 447–478.
- [TP97] D.Turi, G.Plotkin. Towards a mathematical operational semantics, *12<sup>th</sup> LICS*, IEEE, Computer Science Press, 1997, 280–291.
- [UV99] T.Uustalu, V.Vene. Primitive (co)recursion and course-of-value (co)iteration, categorically, *Informatika (IMI, Lithuania)* **10**(1), 1999.
- [UVP01] T.Uustalu, V.Vene, A.Pardo. Recursion Schemes from Comonads, *Nordic Journal of Computing* **8**, 2001, 366–390.