Centrum voor Wiskunde en Informatica

REPORTRAPPORT

SEN

Software Engineering

Software ENgineering

Coalgebraic reasoning in Coq: bisimulation and λ-coiteration scheme

M. Niqui

REPORT SEN-R0806 OCTOBER 2008

Centrum Wiskunde & Informatica (CWI) is the national research institute for Mathematics and Computer Science. It is sponsored by the Netherlands Organisation for Scientific Research (NWO).
CWI is a founding member of ERCIM, the European Research Consortium for Informatics and Mathematics.

CWI's research has a theme-oriented structure and is grouped into four clusters. Listed below are the names of the clusters and in parentheses their acronyms.

Probability, Networks and Algorithms (PNA)

Software Engineering (SEN)

Modelling, Analysis and Simulation (MAS)

Information Systems (INS)

P.O. Box 94079, 1090 GB Amsterdam (NL)
Kruislaan 413, 1098 SJ Amsterdam (NL)
Telephone +31 20 592 9333
Telefax +31 20 592 4199

ISSN 1386-369X

Coalgebraic reasoning in Coq: bisimulation and λ-coiteration scheme

ABSTRACT

In this work we present a modular theory of the coalgebras and bisimulation in the intensional type theory implemented in Coq. On top of that we build the theory of weakly final coalgebras and develop the λ-coiteration scheme, thereby extending the class of specifications definable in Coq. We provide an instantiation of the theory for the coalgebra of streams and show how some of the productive specifications violating the guardedness condition of Coq can be formalised using our library.

2000 Mathematics Subject Classification: 68Q60 ; 18C50
1998 ACM Computing Classification System: F.3.1
Keywords and Phrases: theorem proving; coalgebras; type theory

Coalgebraic Reasoning in Coq: Bisimulation and λ-Coiteration Scheme

Milad Niqui [⋆]

Department of Software Engineering
Centrum Wiskunde & Informatica, The Netherlands
M.Niqui@cwi.nl

Abstract. In this work we present a modular theory of the coalgebras and bisimulation in the intensional type theory implemented in *Coq*. On top of that we build the theory of weakly final coalgebras and develop the λ-coiteration scheme, thereby extending the class of specifications definable in *Coq*. We provide an instantiation of the theory for the coalgebra of streams and show how some of the productive specifications violating the guardedness condition of *Coq* can be formalised using our library.

1 Introduction

Coinduction is a method for proving properties of infinite objects such as streams and infinite trees. It is dual to the usual approach of using induction both for computation and reasoning and can be studied from a category theoretical [15] or type theoretical point of view [10]. Coinduction provides a verification paradigm for programs written in a lazy functional programming e.g. *Haskell*, and hence it is is implemented in many theorem proving tools. Among the many tools used for coinductive reasoning are the ones based on constructive type theory such as *Coq* and *Agda* where *coinductive types* serve this purpose.

The invocation of coinductive types in these tools is quite similar (at least on surface) to the functional programming syntax, these types are defined using their constructors akin to the general recipe for defining algebraic data types. However, type theories where termination is crucial for finite objects enforce similar restriction for ensuring *productivity*[1]. These restrictions are syntactic tests and will inevitably exclude some legitimate productive definitions. Thus not all *Haskell* programs, even those describing total functions, are accepted in coinductive type theories. Several workarounds exists such as [8] where topological properties of fixed points are used or [4] advanced type theoretic techniques are used.

One approach that is understudied is the direct formalisation of various categorical schemes from the theory of coalgebras. The present work indicates that this is a relatively low-cost and generic approach and by formalising a single

[⋆] Research supported by the Netherlands Organisation for Scientific Research (NWO).

[1] Productive functions are those functions on infinite objects that produce provably infinite output.

scheme a large class of total functions can be programmed in coinductive type theory. The scheme we choose is the λ-coiteration scheme of Bartels [2] which is one in a family of schemes intended to expand the basic iteration scheme of final coalgebras. This is a *scheme* i.e., it allows the formalisation of a class of specifications (or *Haskell*-like programs) satisfying a specific syntactic form.

Coalgebraic semantics is so close to coinductive types that in many situations the proof techniques are identical, making coinductive reasoning merely a translation of coalgebras. However in the case of *intensional* type theories this is not the case. In intensional type theory the two objects being provably equal does not entail that they are *convertible*. This restriction is necessary for the decidability of type checking and although it is not a theoretical obstacle for programming, it can be practically inconvenient. In particular formalising category theory is susceptible of this inconvenience, as proving the *uniqueness* of arrows in universal properties of limits adheres to extensional properties of functions. The main workarounds for working extensionally in intensional type theories, is to use *setoids* and work modulo an extensionally defined equality. We will partly follow this through our use of bisimulation equivalence though we will not directly use setoids because we still prefer to benefit from convenient computational properties of intensional equality. The present work shows how one may exploit the intensional equality to the maximum and meanwhile tackle the difficulties through two important tools: (1) working (if necessary) modulo bisimulation equivalence, (2) requiring the functors to satisfy some sort of extensionality. Note that (1) is specific to coalgebras while (2) is applicable to general categorical constructions.

We use the machinery of *Coq* proof assistant to present the formalisation, but the article is applicable to other intensional incarnations of coinductive types (such as *Agda*). Coinductive types themselves will only be used in Section 7 when we instantiate the theory. Throughout the article we use a syntax loosely based on *Coq* syntax, adapted for presenting in an article. In particular we use the uncurried version of the functions when they are presented in mathematical formulae. A complete *Coq* formalisation of the material in this paper can be found in [14].

Related Work. Hancock and Setzer develop the weakly final coalgebra and bisimulation for a very powerful functor capable of representing interactive IO programs in intensional type theory [10]. Their work is formalised in *Agda* by Michelbrink [13] but the latter formalisation uses inductive–recursive universe which is beyond the **CIC**. Neither of [10, 13] study various definition schemes but their work is so expressive that a development of schemes for their functor would considerably extend the class of specifications definable in *Coq*. Our work can be seen as a first step in the formalisation of [10, 13] in *Coq* while along the way extending the class of definition for basic polynomial functors such as streams.

The work by Bertot and Komendantskaya [4] and more recently in [5] uses advanced type theoretic techniques to bypass the guardedness condition of *Coq* for a larger class of functions than those covered by syntactic schemes, including partial functions. Especially in [5] the stream functions that we define in Examples 1–4 (Section 7) are formalised in *Coq* by viewing them as functions on natural numbers and using structural recursion.

2 Coinductive Types

The *Coq* proof assistant [6] is an implementation of the Calculus of Inductive Constructions (**CIC**) extended with coinductive types. Coinductive types were added to *Coq* by Giménez [9]. Their implementation follows the same philosophy as that of inductive types in **CIC**, namely there is a general scheme that allows for formation of coinductive types if their constructors are given, and if these constructors satisfy a strict positivity condition. For example, the type of streams of elements of a set A can be defined using [2] its constructor `Cons` as

```
CoInductive Streams (A : Set) : Set :=
| Cons : A → Streams A → Streams A.
```

From now on we shall use A^ω to denote the type `Streams A`.

After a coinductive type is defined one can introduce its inhabitants and functions into it. Such definitions are given by a *cofixed point* operator cofix . This operator, when given a well-typed definition that satisfies a *guardedness condition*, will introduce an inhabitant of the coinductive type. Assuming that I is a coinductive type, when defining a function $f\colon T \longrightarrow I$ this condition requires each recursive occurrence of f in the body of f should be the immediate argument of a constructor of some inductive or coinductive type [7, 9]. Finally there is a reduction (in fact expansion) rule corresponding to the cofix operator that allows the expansion of a cofixed point only when a case analysis of the cofixed point is done.

Like other syntactic criteria, the guardedness condition of *Coq* excludes some productive functions. An example is the function *convolution* on streams of numbers defined as:

$$(x :: xs) \otimes (y :: ys) := \quad x \cdot y \quad :: \quad (xs \otimes (y :: ys)) \;\oplus\; ((x :: xs) \otimes ys) \tag{2.1}$$

with $\oplus$ being the pointwise addition:

$$(x :: xs) \oplus (y :: ys) := \quad x + y \quad :: \quad xs \oplus ys$$

As evident from the name this function is very useful in formalisation of power series as it corresponds to lazy computation of the coefficients of the convolution product of two power series [12]. Moreover, symbolically it corresponds to the

[2] Note that, as it is the case with algebraic and inductive data types, the type `Stream` and its constructor `Cons` are defined simultaneously.

derivation of formal power series and plays an important role in the stream calculus of [16]. We give a *Coq* formalisation of this function in Section 7. We do this by developing λ-coiteration scheme of [3] that when applied to streams is completely definable in terms of cofix . This is due to the fact that the guardedness condition captures the coiteration scheme of weakly final coalgebras [10]. However, we do not restrict ourselves to streams; we take a more generic approach that is reusable for other similar functors.

3 Extensional Functors and Coalgebras

We are interested in endofunctors on Set[3]. For this we need a type of Set-functors inhibiting a dependent pair[4] of operations $F\colon \mathsf{Set} \longrightarrow \mathsf{Set}$ (on objects) and $l_{F_{X,Y}}: (X{\to}Y){\to}F(X){\to}F(Y)$ (on arrows) satisfying the following properties (ignoring the subscripts X, y in l_F when there is no risk of confusion).

---

l_F_id: $\forall X\ \ (x\colon\ \ F(X)),\ \ x\ \ =\ \ l_F\ \ (\lambda z.z)\ \ x.$

l_F_compose: $\forall XYZ\ \ (g{:}X{\to}Y)\ \ (f{:}Y{\to}Z)\ \ x,$
$$(\lambda z.l_F\ \ f\ \ (l_F\ \ g\ \ z))\ \ x\ \ =\ \ l_F\ \ (\lambda z.f\ \ (g\ \ z))\ \ x.$$

l_F_ext: $\forall XY\ \ (f\ \ g{:}X{\to}Y)\ \ x,\ \ (\forall z, f\ \ z = g\ \ z)\ \to\ l_F\ \ f\ \ x\ \ =\ \ l_F\ \ g\ \ x.$

The first two conditions are standard functorial properties; while l_F_ext will be very helpful in dealing with extensional properties of functor compositions. We shall call a functor satisfying l_F_ext an *extensional functor*. Obviously if one is working in a setting where *functional extensionality* holds i.e.,

$$\forall XY(fg : X{\to}Y), (\forall z, fz = gz) \to f = g\ \ , \qquad (\textsc{Ext})$$

then all functors, in fact all operations on sets, are extensional. So in **CIC** +Ext trivially all functors are extensional. But in **CIC** this is not the case. It is unclear whether the assumption that all functors are extensional is a weaker axiom than Ext, but we can prove that assuming extensionality of some specific functors is tantamount to Ext in the presence of η which is the rule:

$$\forall XY(f : X{\to}Y), \lambda z.(fz) = f\ \ . \qquad (\eta)$$

Proposition 1. *Assume η. For a given set A the functor $F(X) := X^A$ is extensional if and only if all function $X{\to}A$ satisfy* Ext.

Proof. ($\Leftarrow$) is trivial, for ($\Rightarrow$) assume l_F_ext and let $f, g\colon A \longrightarrow A$ be given s.t.

$$\forall z, fz = gz\ \ . \qquad (3.1)$$

Then by applying $l_{F_{A,X}}$_ext to f, g, $x := \lambda z.z\colon F(A)$ and (3.1) we have $\lambda z.fz = \lambda z.gz$. Now by ($\eta$) we obtain $f = g$, so f and g satisfy Ext. □

[3] This is the type Set of *Coq* which corresponds to constructive sets and computations. It can be identified with any of categorical models of type theory.

[4] In fact these are formalised as *module types* in *Coq* (see [14].

It is well-known that **CIC**+η is weaker than **CIC**+Ext and hence the above shows that $F(X) := X^A$ cannot be proven to be an extensional functor inside **CIC**. On the other hand each functor composed of finite sums and products seem to be extensional in **CIC**. In fact we can prove the following lemma in **CIC**.

Lemma 1. *(i) The constant functor, sending every set to a fixed set U an each arrow to the identity on U is extensional.*
(ii) The identity functor, identity on objects and arrows, is extensional.
(iii) Disjoint sum of two extensional functors obtained by case analysis on arrows is extensional.
(iv) Product of two extensional functors obtained by pairing on arrows is extensional.
(v) Composition of two extensional functors obtained by composition on arrows is extensional.

The proof can be found in [14] and ensures the extensionality of most polynomial functions used in practice bar those based on exponential. In particular it holds for functors used in Examples 1–4.

The main advantage of l_F_ext is that it eliminates the need for Ext without having to resort to setoids functors and hence leaving us with a light weight formalisation. This is in contrast with the formalisation of category theory in [11] where setoids are used.

After this we define the notion of F-coalgebra for extensional functors as a set together with a transition structure.

```
Record F_coalg : Type :=
{ st : Set
; tr : st → F st
}.
```

Then we need to define the lifting of a relation R on the image of functor F; this will later be used to for expressing the commutativity of diagrams involving the weakly final coalgebra.

```
Definition l_Rel(F) (S₁ S₂ : F_coalg) (R : S₁.st→S₂.st→Prop)
                              (z_x: F S₁.st) (z_y: F S₂.st) : Prop :=
                         ∃ x y, R x y ∧ z_x = S₁.tr x ∧ z_y = S₂.tr y.
```

4 Bisimulation

Bisimulation is the basic tool for studying the elements in a coalgebra. First we recall the usual categorical definition of F-*bisimulation* [15]: given two sets X, Y and a relation $R \subseteq X \times Y$ is a bisimulation between X and Y if there is a map

$\gamma\colon R \longrightarrow F(R)$ s.t. both squares in this diagram commute (by π_{ij} we denote the ith projection of a j-tuple):

$$\begin{tikzcd}
X \arrow[d, "\alpha_X"'] & R \arrow[l, "\pi_{12}"'] \arrow[r, "\pi_{22}"] \arrow[d, dotted, "\gamma"] & Y \arrow[d, "\alpha_Y"] \\
F(X) & F(R) \arrow[l, "F\pi_{12}"] \arrow[r, "F\pi_{22}"'] & F(Y)
\end{tikzcd}$$

In **CIC** though, where we use dependent types for subsets, there is a distinction between a Prop-valued relation $R\colon X{\to}Y{\to}\mathsf{Prop}$ and the set of pairs in $\{(x,y) \in X \times Y | Rxy\}$. The latter is a set of dependent pairs also called a Σ-type. Because we will be composing Σ-types built on a relation in Prop with Σ-types built on other Σ-types we need to fix the notation. By $\{\Sigma x\colon X, \phi(x)\}$ we denote the set of elements of X satisfying $\phi : X{\to}\mathsf{Prop}$, and by $\{\overline{\Sigma} x\colon X, f(x)\}$ we denote the set of elements X for which $f(x)$ is inhabited (here $f\colon X{\to}\mathsf{Set}$ is an X-indexing of sets). We shall use variable R for Prop-valued and variable ρ for Set-valued ones, i.e., $\rho\colon X{\to}Y{\to}\mathsf{Set}$. Given a relation R (resp. ρ) we write $\{\Sigma(R)\}$ (resp. $\{\overline{\Sigma}(\rho)\}$ as a shorthand for $\{\Sigma u\colon X{\times}Y,\ R\ \ \pi_{12}(u)\ \ \pi_{22}(u)\}$ (resp. $\{\overline{\Sigma} u\colon X\times Y,\ \rho\ \ \pi_{12}(u)\ \ \pi_{22}(u)\}$). Note that an element of $\{\Sigma(R)\}$ is a 3-tuple consisting a pair from $X \times Y$ and a proof that they satisfy R hence we use π_{i3} to access to projections.

With the above notation an F-bisimulation for an extensional functor F will be determined be the existence of $\gamma\colon \{\Sigma(R)\}{\to}F\{\Sigma(R)\}$. Set theoretically this is equivalent to $\gamma\colon R{\to}F(R)$ but in **CIC** the distinction is necessary. But this is not the only discrepancy: The above diagram for bisimulation is an existential statement. In order to formalise the existence of the γ in a way that can be later used as a witness, in **CIC** we have to define the *set* of all F-bisimulations between X and Y.

Following the above we define two predicates; first when a Prop-valued relation is bisimulation and second for a Set-valued relation:

$\texttt{Definition}$ is_{F_bisim} $(S_1\ \ S_2\colon F\_\mathsf{coalg})$ $(R\colon S_1.st{\to}S_2.st{\to}\mathsf{Prop})$ $:\ =$

$$\Big\{\overline{\Sigma}\gamma\colon \{\Sigma R\}{\to}F\{\Sigma R\},\ \ \forall y\colon \{\Sigma R\},$$
$$l_F\ \ \pi_{13}\ \ \gamma(y) = S_1.tr\ \ (\pi_{13}(y))\ \ \bigwedge\ \ l_F\ \ \pi_{23}\ \ \gamma(y) = S_2.tr\ \ (\pi_{23}(y))\Big\}.$$

$\texttt{Definition}$ $is_{F\_\sigma bisim}$ $(S_1\ \ S_2\colon F\_\mathsf{coalg})$ $(\rho\colon S_1.st{\to}S_2.st{\to}\mathsf{Set})$ $:\ =$

$$\Big\{\overline{\Sigma}\gamma\colon \{\overline{\Sigma}\rho\}{\to}F\{\overline{\Sigma}\rho\},\ \ \forall y\colon \{\overline{\Sigma}\rho\},$$
$$l_F\ \ \pi_{13}\ \ \gamma(y) = S_1.tr\ \ (\pi_{13}(y))\ \ \bigwedge\ \ l_F\ \ \pi_{23}\ \ \gamma(y) = S_2.tr\ \ (\pi_{23}(y))\Big\}.$$

We usually ignore the first two arguments of is_{F_bisim} and $is_{F\_\sigma bisim}$ and simply write $is_{F\_bisim}(R)$. Now we can define when a bisimulation is maximal.

$\texttt{Definition}$ $is^{max}_{F\_bisim}$ $(S_1\ \ S_2 : F\_\mathsf{coalg})$ $(R : S_1.st{\to}S_2.st{\to}\mathsf{Prop})$ $:=$

$$is_{F\_bisim}(R)\ \ \wedge\forall\rho,\ \ is_{\sigma F\_bisim}(\rho){\to}\forall s_1 s_2,\ \ \rho\ \ s_1\ \ s_2{\to}R\ \ s_1\ \ s_2.$$

As we will see later the subtle occurrence of a Set-valued relation ρ is crucial in the proof of the fact that bisimulation is closed under the composition.

It is well-known that the maximal bisimulation between any two F-coalgebras exist [15]. In our theory we assume the existence of a maximal bisimulation. Later on for each concrete functor we have to build a concrete relation which should be proven to satisfy $is^{max}_{F\_bisim}$. This can always be built using as a coinductive type [10], as we shall see for streams in Section 7.

It is known that for bisimulation to be closed under composition functor F should satisfy some additional property, e.g. in [15] F is required to preserve weak pullbacks. We require a similar albeit weaker restriction. First we define the carrier set of the weak pullback of $f\colon X \longrightarrow Z$ and $g\colon Y \longrightarrow Z$ to be the set $\mathsf{WP}(f,g) := \{\Sigma u\colon X \times Y, f(\pi_{12}(u)) = g(\pi_{22}(u))\}$. Subsequently we require that a function $i_{wpF}\colon \mathsf{WP}(l_F(f), l_F(g)) \longrightarrow F(\mathsf{WP}(f,g))$ satisfying the following property exist.

$$\begin{aligned}\mathsf{WP}^{\rightarrow}_{F} :\ \ \forall XYZ\ \ (f : X{\rightarrow}Z)\ \ (g : Y{\rightarrow}Z)\ \ (u : \mathsf{WP}(l_F(f), l_F(g))),\\ l_F\ \ \pi_{13}\ \ i_{wpF}(u) = \pi_{13}(u)\ \ \bigwedge\ \ l_F\ \ \pi_{23}\ \ i_{wpF}(u) = \pi_{23}(u).\end{aligned}$$

Evidently this is weaker than the assumption that F preserves weak pullbacks because we only require the preservation of the pullback arrows, and even then up to the existence of a one-way map i_{wpF} which is not required to be an isomorphism. In fact it seems that in **CIC** one cannot prove the stronger assumption

$$\mathsf{WP}(l_F(f), l_F(g)) = F(\mathsf{WP}(f,g)) \tag{4.1}$$

even for simple polynomial stream functor $F(X) := B \times X$. The reason is that (4.1) for streams would state the equality between a Σ-type and a non-dependent pair of a set and another Σ-type and hence a proof would require the commutativity of the two constructors of inductive types for pairing and dependent pairing. This is impossible to do in **CIC** without additional axioms. In contrast we can prove $\mathsf{WP}^{\rightarrow}_{F}$ for stream functor; thus the relaxation in weak pullback preservation condition is a necessity, in other words the stream functor in **CIC** is a functor for which the bisimulation is closed under composition and for which we cannot prove that it preserves weak pullbacks.

Given the above requirements, i.e., a maximal F-bisimulation between coalgebras S_1 and S_2 and a map i_{wpF} satisfying $\mathsf{WP}^{\rightarrow}_{F}$ we can develop a theory of bisimulation. First we need the following properties [5].

Lemma 2. *i)* $is_{F\_bisim}(S_1, S_2, R) \implies is_{\sigma F\_bisim}(S_1, S_2, R)$.
ii) $is_{F\_bisim}(S, S, =)$, *i.e., propositional equality is a bisimulation relation.*
iii) $is_{F\_bisim}(S_1, S_2, R) \implies is_{F\_bisim}(S_2, S_1, \lambda xy.Ryx)$.
iv) $is_{F\_bisim}(S_1, S_2, R_{12}) \wedge is_{F\_bisim}(S_2, S_3, R_{23}) \implies$
$is_{\sigma F\_bisim}\big(S_1, S_3, \lambda xz.\{\Sigma y, R_{12}xy \wedge R_{23}yz\}\big)$, *i.e., bisimulation preserves composition.*

[5] This theorem and all the following ones are all formalised in *Coq* and available in [14].

The only technical part of the proof is part (iv). The relation $R_{12}\bar{\circ}R_{23} := \lambda xz.\{\Sigma y, R_{12}xy \wedge R_{23}yz\}$ is the counterpart of the set-theoretic composition of two relations $\lambda xz.\exists y, R_{12}xy \wedge R_{23}yz$. For the rest we follow the proof in [15], defining the maps in the following diagram. Here $X := \mathsf{WP}(\pi_{22}^{R_{12}}, \pi_{22}^{R_{23}})$ i.e., the weak pullback for $\pi_{22}^{R_{ij}} \colon \{\Sigma(R_{ij})\} \longrightarrow S_2$ and $\pi_{(1,2)3}$ is the function mapping a 3-tuple to its first two elements.

$$\begin{tikzcd}
 & & \{\overline{\Sigma}(R_{12}\bar{\circ}R_{23})\} \arrow[lld, bend right, "\pi_{14}"'] \arrow[rrd, bend left, "\pi_{24}"] \arrow[d, "\imath"] & & \\
S_1.st \arrow[d, "S_1.tr"'] & \{\Sigma(R_{12})\} \arrow[l, "\pi_{12}"'] \arrow[d, "\gamma_{12}"'] & X \arrow[l, "\pi_{(1,2)3}"'] \arrow[r, "\pi_{(1,2)3}"] \arrow[d, dotted, "X.tr"] & \{\Sigma(R_{23})\} \arrow[r, "\pi_{22}"] \arrow[d, "\gamma_{23}"'] & S_3.st \arrow[d, "S_3.tr"] \\
F(S_1.st) & F\{\Sigma(R_{12})\} \arrow[l, "F\pi_{12}"] & F(X) \arrow[l, "F\pi_{(1,2)3}"] \arrow[r, "F\pi_{(1,2)3}"'] \arrow[d, "F\jmath"] & F\{\Sigma(R_{23})\} \arrow[r, "F\pi_{22}"'] & F(S_3.st) \\
 & & F\{\overline{\Sigma}(R_{12}\bar{\circ}R_{23})\} \arrow[llu, bend left, "F\pi_{12} \circ \pi_{(1,2)3} \circ \imath"] \arrow[rru, bend right, "F\pi_{22} \circ \pi_{(1,2)3} \circ \imath"'] & &
\end{tikzcd}$$

In this diagram $\jmath$ is the map sending an element $\langle s_1, s_2, \phi_{12}, s_2', s_3, \phi_{23}, \phi_=\rangle$ of X to $\langle s_1, s_3, s_2, \phi_{123}\rangle$, where ϕ's are proof obligations and in particular $\phi_=$ is a proof that $s_2 = s_2'$. Likewise $\imath$ is the 'inverse' of $\jmath$ and

$$\imath\langle s_1, s_3, s_2, \phi_{123}\rangle := \langle s_1, s_2, \phi_{12}, s_2, s_3, \phi_{23}, \phi_{\mathrm{refl}}\rangle \ .$$

The main part is defining a coalgebraic structure on X to obtain the transition map $X.tr$. For this we use the map $p \colon X \longrightarrow \mathsf{WP}(l_F(\pi_{22}^{R_{12}}), l_F(\pi_{22}^{R_{23}}))$ defined as

$$p\langle s_1, s_2, \phi_{12}, s_2', s_3, \phi_{23}, \phi_=\rangle := \langle \gamma_{12}\langle s_1, s_2, \phi_{12}\rangle, \gamma_{23}\langle s_2', s_3, \phi_{23}\rangle, \phi_{l_F}\rangle$$

where ϕ_{l_F} is the proof that

$$l_F(\pi_{22}^{R_{12}})\big(\gamma_{12}\langle s_1, s_2, \phi_{12}\rangle\big) = l_F(\pi_{22}^{R_{23}})\big(\gamma_{23}\langle s_2', s_3, \phi_{23}\rangle\big) \ ,$$

and is obtained by $\phi_=$ and the commutativity of the bisimulation diagrams for $\{\Sigma(R_{12})\}$ and $\{\Sigma(R_{23})\}$. Now taking $X.tr := i_{wpF} \circ p$ we can prove that $X.tr$ is indeed a homomorphism of coalgebras i.e., the small squares in the above diagram commute. Subsequently the entire diagram above commutes. Which means $F\jmath \circ X.tr \circ \imath$ is the map making $\{\overline{\Sigma}(R_{12}\bar{\circ}R_{23})\}$ an F-bisimulation and thus completing the proof.

Using Lemma 2 we can easily derive the following theorem.

Theorem 1. *For any coalgebra S the maximal bisimulation on S is an equivalence relation.*

Theorem above is the main tool for a generic definition of bisimulation as an extensional equality on coalgebras: due to our modular formalisation in **CIC**, each time we instantiate the theory of this section with an extensional Set-functor satisfying $\mathsf{WP}_F^{\rightarrow}$ and a maximal bisimulation relation we get this theorem for free.

As a final remark we note that all the machinery based on weak pullbacks and $\overline{\Sigma}$-types is necessitated by the proof of transitivity which in turn is based on Lemma 2.iv. In other words, the reflexivity and the symmetry of maximal bisimulation holds for any extensional functor and for the weaker notion maximality obtained by replacing $is_{\sigma F\_bisim}$ by is_{F_bisim} in the definition of $is^{max}_{F\_bisim}$.

5 Weakly Final Coalgebras

Continuing the set-up so far we assume F is a weak pullback preserving extensional functor so that the bisimulation theory of the previous section is derivable. First we define when a coalgebra is *weakly final*:

$\texttt{Definition}\ is_{F\_wfin}\ (S_0\colon F\_\texttt{coalg})\ :=\ \forall\ (S_1\colon F\_\texttt{coalg}),$
$\{\Sigma\texttt{unfld}_F\colon S_1.st{\to}S_0.st, \forall s_1,\ \ S_0.tr\ \ (\texttt{unfld}_F\ \ s_1) = l_F\ \ \texttt{unfld}_F\ \ (S_1.tr\ \ s_1)\}.$

If Ω satisfies the above property we call the maximal bisimulation on Ω the *bisimilarity* and we denote it by $\cong$. According to the above definition the existence of a coalgebra homomorphism originating from any other coalgebra is enough. For concrete functors $S_0.st$ can be taken to be a suitably chosen coinductive type with $S_0.tr$ being the inverse of the constructors. Though, in each concrete case we cannot prove the uniqueness of $\texttt{unfld}_F(S_1)$ without assuming EXT as an axiom. But fortunately we can prove the following property assuming $is_{F\_wfin}(\Omega)$ holds.

$$\begin{aligned}\Omega_{\text{unique}}\colon \forall\ \ (S\colon F\_\texttt{coalg})\ \ (f\ \ g\colon S.st{\to}\Omega.st),&\\ \forall s_0,\ \ \Omega.tr\ \ (f\ \ s_0)\ \ =\ \ l_F(f)\ \ (S.tr\ \ s_0)\ \ )\ \to&\\ \forall s_0,\ \ \Omega.tr\ \ (g\ \ s_0)\ \ =\ \ l_F(g)\ \ (S.tr\ \ s_0)\ \ )\ \to\ \forall s,\ \ f\ \ s\ \ \cong\ \ g\ \ s.&\end{aligned}$$

Finally, we need another requirement that is needed when proving commutativity of diagrams up to bisimilarity (cf. Section 6).

$$\begin{aligned}l^{\cong}_F\text{\_ext}\colon \forall X\ \ (f\ \ g : X{\to}\Omega.st)\ \ (y\colon F\ \ X),\ \ \big(\forall\ \ x,\ \ f(x)\ \ \cong\ \ g(x)\big)\ \to&\\ l_{Rel(F)}\big(\Omega,\Omega,\cong,\ \ l_F(f)(y),\ \ l_F(g)(y)\big).&\end{aligned}$$

Note that here we take as argument arbitrary set X which does not need to have a coalgebraic structure. It allows us to use this property in more general situations, e.g. in next section we use this on a bi-algebraic structure.

So far we have always used the (intensional)[6] propositional equality to use the commutativity of diagrams. However it is well-known that for weakly final coalgebras the natural equality is the bisimilarity [10] which can be used

[6] Propositional equality of **CIC** in the empty context is indistinguishable from the intensional equality of the conversion rules of the type theory [1].

for proofs based on *coinduction principle*. The coinduction principle states that maximal bisimulation is the equality. In **CIC** this may be stated as 'the maximal bisimulation on weakly final coalgebra is propositional equality', but it is not provable. I.e., for concrete functors we cannot prove that $\cong$ and $=$ coincide. But given Theorem 1 we know that any weakly final coalgebra can be turned into a setoid with a corresponding coinduction proof principle. And thus, finding bisimulation will result in equality in that setoid. This enables us to translate and verify in **CIC** the proofs by coinduction principle.

6 λ-Coiteration Scheme

Our theory so far has the *coiteration scheme* which is the existence of the arrow in is_{F_wfin}. The scheme of λ-coiteration was developed in order to extend the class of *Haskell*-like specifications beyond coiteration [3]. Our purpose is to develop the λ-coiteration scheme inside **CIC** in the theory of previous sections.

First we sketch the scheme given in [3]. Let B, T be two extensional functors and Ω be a weakly final B-coalgebra. Let $\Lambda \colon TB \Longrightarrow BT$ be a natural transformation. Given a map $g \colon X \longrightarrow BTX$ if the diagram below commutes then f is called λ-*coiterative arrow induced by* g.

$$\begin{tikzcd}
X \arrow[rr, dotted, "f"] \arrow[d, "g"'] & & \Omega.st \arrow[d, "\Omega.tr"] \\
BT(X) \arrow[r, dotted, "BT(f)"'] & BT(\Omega) \arrow[r, "B(\beta)"'] & B(\Omega)
\end{tikzcd}$$

Here $\beta := \pi_{12}(\phi \quad S_0)$ where ϕ is a proof of $is_{F\_wfin}(\Omega)$ and S_0 the coalgebra with carrier $T(\Omega)$ and transition function $\Lambda_\Omega \circ T(\Omega.tr) \colon T(\Omega) \longrightarrow BT(\Omega)$.

In [3] it is proven that if the ambient category possesses countable coproducts then given g, Λ a unique λ-coiterative arrow exists. In **CIC** countable coproducts is an $\mathbb{N}$-indexed family of sets and always exists (see H below) , and thus we can prove the existence of λ-coiterative arrow for B and T without further assumptions.

Our proof follows the one in [3] with some simple modifications with respect to equality. For presenting the λ-coiterative arrow we need to formalise several structures of [3] in **CIC**. The translation of these structures is straightforward. Let $H := \lambda X.\{\overline{\Sigma} j \colon \mathbb{N}, \{\Sigma x \colon T^j(X), \top\}\}$ where T^j is the recursively defined j-th iteration of T and $\top$ is the universally true proposition. We can prove by induction that T^j is an extensional Set-functor. On the other hand for H we do not need the extensionality and it seems that it is not provable either.

For each j and any set Y with $y \in T^j(Y)$ let

$$\begin{aligned} \imath_{jY} &\colon T^j(Y) \longrightarrow H(Y) \\ \imath_{jY}(y) &:= \langle j, y, \top \rangle \ . \end{aligned}$$

Furthermore, for a sets Y, Z and $\mathbb{N}$-indexed family of functions $f_j \colon T^j Y \longrightarrow Z$ let $[f_j]_0^\infty \colon H(Y) \longrightarrow Z$ be

$$[fj]_0^\infty := \lambda x. f_{\pi_{13}(x)}(\pi_{23}(x)) \enspace .$$

Next let $\chi_X := [\imath_{(j+1)X}]_0^\infty$. We define the iteration of $\mathit{\Lambda}$ recursively as:

$$\begin{aligned} \mathit{\Lambda}_X^0 &= \lambda x.x \\ \mathit{\Lambda}_X^{j+1} &= \lambda x \colon T^j(TB(X)).\mathit{\Lambda}_{T(X)}(l_{T^j} \quad \mathit{\Lambda}_X \quad x) \enspace . \end{aligned}$$

Finally let $\mathit{\Lambda}_X^* \colon HB(X) \longrightarrow BH(X)$ be

$$\mathit{\Lambda}_X^* := [\lambda x \colon T^j(X).\, l_B \quad \imath_{jX} \quad (\mathit{\Lambda}_X^j(x))]_0^\infty \enspace .$$

Now we can define the function making the above diagram commute.

Definition 1. *Given $\mathit{\Lambda}$ and g as above let S_1 be the coalgebra with carrier $H(X)$ and the transition function*

$$\lambda x \colon H(X).\, l_B \quad \chi_X \quad \big(\mathit{\Lambda}_{T(X)}^*(l_H \quad g \quad x)\big) \enspace .$$

Let $h := \pi_{12}(\phi \quad S_1)$ be the map given by weak finality of $\mathit{\Omega}$ (where ϕ is a proof of $is_{F\_wfin}(\mathit{\Omega})$). Then we define $coit_{\mathit{\Lambda}Xg} \colon X \longrightarrow \mathit{\Omega}.st$ as

$$coit_{\mathit{\Lambda}Xg} := \lambda x \colon X.\, h\big(\imath_{0X}(x)\big) \enspace .$$

In our setting we should state the commutativity using bisimilarity.

Theorem 2. *The map $coit_{\mathit{\Lambda}Xg}$ is the* λ-coiterative arrow induced by *g up to bisimilarity, i.e.,*

$$l_{Rel(F)}\Big(\mathit{\Omega}, \mathit{\Omega}, \cong, \mathit{\Omega}.tr\big(coit_{\mathit{\Lambda}Xg}(x)\big), l_B \quad \beta \quad (l_B \quad (l_T(coit_{\mathit{\Lambda}Xg}) \quad (g(x)))\Big) \enspace .$$

The proof of Theorem 2 is quite technical and can be found in the *Coq* formalisation[14]. It follows to a great deal the paper proof in [3]. However working in **CIC** results in some minor differences. As we mentioned above in **CIC** we have H for free, on the other hand we have to explicitly assume that the functor B is extensional and satisfies $l_B^{\cong}$_ext. More technical difference is that for each j we need a map $!_{jX} \colon T^j(T(X)) \longrightarrow T(T^j(X))$ recursively defined as

$$\begin{aligned} !_{0X} &= \lambda x.x \\ !_{(j+1)X} &= !_{jT(X)} \enspace . \end{aligned}$$

The role of this map is to replace the reasoning steps that rely on the intensional equality $T^j(T(X)) = T(T^j(X))$. This is because although this equality is provable in **CIC** as a propositional equality the two sides when considered as types are not convertible[7]. Such non-convertibility would be an obstacle in proving the commutativity of diagrams by naturality laws, which are otherwise automatically proven by the conversion mechanism of *Coq*. As we mentioned using $!_{jX}$ is a workaround that, although making proofs more tedious, works suitably.

7 Streams

In this section we show that the the theory of Sections 3–6 can be instantiated by the important case of streams, and hence the requirements that we put on functors are reasonable. Note that in those section we did not use coinductive types, while in this section we will use the coinductive types of *Coq*.

Fix a set B. Already from Lemma 1 we know that the stream functor defined as $F(X) := B \times X$ with $l_{B\times X}(f)\langle b, x\rangle = \langle b, f(x)\rangle$ is an extensional functor. This allows us to build the coalgebra $B\times$_`coalg` of the functor above with the obvious components of the transition map:

$$\mathrm{hd}_S\colon S.st \to B := \lambda s.\pi_{12}(S.tr(s)) \ , \qquad \mathrm{tl}_S\colon S.st \to S.st := \lambda s.\pi_{22}(S.tr(s)).$$

Now we need a maximal bisimulation between any two $B\times$-coalgebras. This will be a coinductive type defined as:

`CoInductive` $max^{S_1 S_2}_{B\times\_bisim}$ $(s_1 : S_1.st)$ $(s_2 : S_2.st)$: Prop $:=$
| $\quad max_{B\times\_bisim0}$: $\mathrm{hd}_{S_1}(s_1) = \mathrm{hd}_{S_2}(s_2) \ \rightarrow\ max^{S_1 S_2}_{B\times\_bisim}\ \mathrm{tl}(s_1)\ \mathrm{tl}(s_2) \ \rightarrow$
$\qquad\qquad max^{S_1 S_2}_{B\times\_bisim}\ s_1\ s_2.$

Subsequently we can prove this lemma:

Lemma 3. *Let S_1, S_2 be two $B\times$-coalgebras. Then*

$$is^{max}_{B\times\_bisim}(S_1, S_2, max^{S_1 S_2}_{B\times\_bisim})$$

Proof. The proof has two parts. First to prove that $max^{S_1 S_2}_{B\times\_bisim}$ is a bisimulation take

$$\gamma := \lambda x.\langle \mathrm{hd}_{S_1}(\pi_{13}(x)), \langle \mathrm{tl}_{S_1}(\pi_{13}(x)), \mathrm{tl}_{S_2}(\pi_{23}(x)), \phi\rangle\rangle$$

where ϕ is a proof that

$$\langle \mathrm{hd}_{S_1}(\pi_{13}(x)), \mathrm{tl}_{S_1}(\pi_{13}(x))\rangle = S_1.st(\pi_{13}(x)) \ ,$$
$$\langle \mathrm{hd}_{S_1}(\pi_{13}(x)), \mathrm{tl}_{S_2}(\pi_{23}(x))\rangle = S_2.st(\pi_{23}(x)) \ .$$

In the second part for each ρ satisfying $is_{\sigma F\_bisim}(\rho)$ and each s_1, s_2 for which the set $\rho s_1 s_2$ is inhabited, we ought to build an element of the coinductive type $max^{S_1 S_2}_{B\times\_bisim}(s_1, s_2)$. That means we employ the constructor $max_{B\times\_bisim0}$ and

[7] Obviously there are two possible ways to define T^j. No matter which of the two ways we take we will always need $!_{jX}$ or its inverse.

use the facts that $\mathrm{hd}_{S_1}(s_1) = \mathrm{hd}_{S_2}(s_2)$ and $max^{S_1 S_2}_{B\times\_bisim}\big(\mathrm{tl}(s_1), \mathrm{tl}(s_2)\big)$. Both of these are provable using the commutativity of bisimulation for ρ. The latter also uses the fact that

$$\rho \quad \mathrm{tl}_1(s_1) \quad \mathrm{tl}_2(s_2) \quad \neq \quad \emptyset \ .$$

□

Next we define the map $i_{wpB\times}$ as follows (again ϕ's are proof obligations).

$$i_{wpB\times}\langle\langle b_0, s_0\rangle, \langle b_1, s_1\rangle, \phi_{01}\rangle := \langle b_0, \langle s_0, s_1, \phi_{\mathrm{refl}}\rangle\rangle \ .$$

With this definition we can prove $\mathsf{WP}^{\rightarrow}_{B\times}$ and hence the ingredients of the bisimulation theory are all supplied. This means that we get Theorem 1 for free.

At this point we focus on weakly final coalgebra of streams. Consider the coinductively defined set B^w of streams over B introduced in Section 2. Taking $\nu_{B\times} := \langle B^\omega, \langle \mathrm{hd}_{B^\omega}, \mathrm{tl}_{B^\omega}\rangle\rangle$ to be the coalgebra of streams, it is easy to prove that

$$is_{B\times\_wfin}(\nu_{B\times})$$

holds: the witness is the *unfold* map for streams which is easily defined using the cofix operator of *Coq*:

$\texttt{CoFixpoint unfld}_{B\times}$ $(S_1\colon B\times \texttt{\_coalg})$ $(s_1\colon S_1.st)\colon$ $B^\omega :=$
$\qquad\qquad\qquad\qquad \texttt{Cons}\ \ \mathrm{hd}_{S_1}(s_1)\ \ \big(\texttt{unfld}_{B\times}\ \ S_1\ \ \mathrm{tl}_{S_1}(s_1)\big).$

Proving the uniqueness $\nu_{B\times_{\mathrm{unique}}}$ needs the following lemma.

Lemma 4. *Let be a $B\times$-coalgebra. Then*

i) $\texttt{unfld}_{B\times}\ \ S\ \ s = \texttt{Cons}\ \ \mathrm{hd}_S(s)\ \ \texttt{unfld}_{B\times}\big(\mathrm{tl}_S(s)\big)$.

ii) Let $f\colon S.st \longrightarrow B^\omega$ be such that

$$\forall s\colon S.st, f(s) = \texttt{Cons}\ \ \mathrm{hd}_S(s)\ \ f\big(\mathrm{tl}_S(s)\big) \ .$$

Then for all s in S we have

$$\texttt{unfld}_{B\times}\ \ S\ \ s\ \ \cong\ \ f(s) \ .$$

Part (i) is trivial (see definition of $\texttt{unfld}_{B\times}$), while part (ii) uses constructor of the coinductive type $max^{\nu_{B\times}\nu_{B\times}}_{B\times\_bisim}$ and the cofix operator of *Coq* to build the bisimilarity [14].

Finally the proof of $l^{\cong}_{B\times}$_ext is a routine use of properties of $\cong$ as an equivalence relation.

Then we can apply the scheme developed in the previous section to define streams and functions on streams. We illustrate this by some examples. For each example we mention which parameters for the λ-coiteration scheme should be taken. All the choices for functor T in these examples are extensional by Lemma 1. Each example contains a *Haskell*-like specification; applying Theorem 2 and replacing the definition of $l_{Rel(F)}$ enables us to derive the specifications as a bisimilarity.

Example 1. For function *convolution* defined in Section 2 choose:

$$\begin{aligned}
&T := \lambda X.X \times X\\
&\Lambda_X := \lambda x.\pi_{14}(x) + \pi_{34}(x), \langle \pi_{24}(x), \pi_{44}(x)\rangle\\
&g := \lambda x\colon B^\omega \times B^\omega.\langle \mathrm{hd}_{B^\omega}\big(\pi_{12}(x)\big) \cdot \mathrm{hd}_{B^\omega}\big(\pi_{22}(x)\big),\\
&\qquad\qquad\qquad\qquad \langle \mathrm{tl}_{B^\omega}\big(\pi_{12}(x)\big), \pi_{22}(x), \pi_{12}(x), \mathrm{tl}_{B^\omega}\big(\pi_{22}(x)\big)\rangle\rangle \ .
\end{aligned}$$

Then given two streams xs, ys we can define $xs \otimes ys$ as $\mathrm{coit}_{\Lambda X g}\langle xs, ys\rangle$. In this case using Theorem 2 leads to the following bisimilarity for $\otimes$ which is the counterpart of (2.1) in the intensional setting of *Coq*:

$$xs \otimes ys \ \cong\ \texttt{Cons} \ \ \big(\mathrm{hd}_{B^\omega}(xs) \cdot \mathrm{hd}_{B^\omega}(ys)\big) \ \ \big(\ \mathrm{tl}_{B^\omega}(xs) \otimes ys \ \ \oplus \ \ xs \otimes \mathrm{tl}_{B^\omega}(ys)\ \big)$$

Here $xs \oplus ys := \beta\langle xs, ys\rangle$ can also be proven, by Lemma 4.(i), to satisfy

$$xs \oplus ys \ = \ \texttt{Cons} \ \ \big(\mathrm{hd}_{B^\omega}(xs) + \mathrm{hd}_{B^\omega}(ys)\big) \ \ \big(\ \mathrm{tl}_{B^\omega}(xs) \ \ \oplus \ \ \mathrm{tl}_{B^\omega}(ys)\ \big)$$

Note that for $\oplus$ we get an equality, which by Lemma 2.(ii) leads to a bisimilarity. □

Example 2. As a simpler example consider the stream of powers of two from [3] specified as

$$\texttt{pow} := 1 :: \texttt{pow} \oplus \texttt{pow}$$

As shown in [3] `pow` can be defined by λ-coiteration employing the same T and Λ as in Example 1. We merely have to use a different g, this time a coalgebraic structure on the unit set $\mathbf{1} = \{*\}$:

$$\begin{aligned}
&g := \lambda x\!:\!\mathbf{1}.\langle 1, \langle *, *\rangle\rangle\\
&\texttt{pow} := \mathrm{coit}_{\Lambda X g}(*)
\end{aligned}$$

□

Example 3. The stream of natural numbers with the specification

$$\texttt{nats} := 0 :: \mathit{map} \ \ \lambda n.n{+}1 \ \ \texttt{nats}$$

is a well-known example of a stream definition not accepted by the guardedness condition of *Coq* [9]. This is definable in *Coq* by taking:

$$\begin{aligned}
&T := \lambda X.X \times B^B \times X\\
&\Lambda_X := \lambda x.\langle \pi_{13}(x)\big(\pi_{23}(x)\big), \langle \pi_{13}(x), \pi_{33}(x)\rangle\rangle\\
&g := \lambda x\!:\!\mathbf{1}.\langle 0, \langle \lambda n.n{+}1, *\rangle\rangle\\
&\texttt{nats} := \mathrm{coit}_{\Lambda X g}(*)
\end{aligned}$$

□

Example 4. The Fibonacci specification studied in [5] can also be defined using the λ-coiteration scheme but the specification should be slightly unfolded as

$$\texttt{fibs} := 0 :: \ \oplus_3 (1, \texttt{fibs}, \texttt{fibs}) \tag{7.1}$$

where $\oplus_3$ is a ternary unfolding of $\oplus$:

$$\oplus_3(x_0,\ x :: xs,\ y :: ys) := x_0 + y :: \ \oplus_3 (x, xs, ys)$$

We define this by taking

$$\begin{aligned}
&T := \lambda X.B \times X \times X \\
&\Lambda_X := \lambda x.\langle \pi_{15}(x) + \pi_{45}(x), \langle \pi_{25}(x), \pi_{35}(x), \pi_{55}(x)\rangle\rangle \\
&g := \lambda x{:}\mathbf{1}.\langle 0, \langle 1, *, *\rangle\rangle \\
&\texttt{fibs} := \mathrm{coit}_{\Lambda X g}(*)
\end{aligned}$$

Again Theorem 2 gives us (7.1) up to bisimilarity. Furthermore we can prove the following bisimilarity which corresponds to the specification used in [5] as a definition of stream of Fibonacci numbers.

$$\texttt{fibs} \cong \texttt{Cons} \ \ 0 \ \ \big(\texttt{Cons} \ \ 1 \ \ (\ \mathrm{tl}_{B^\omega}(\texttt{fibs}) \ \oplus \ \texttt{fibs}\ )\big) \ .$$

Note that we are using $\oplus$ from Example 1. The proof of this bisimilarity is based on the following properties of $\oplus$ and $\oplus_3$.

$$\begin{aligned}
&\oplus_3 (x, xs, ys) \cong (\texttt{Cons} \quad x \quad xs) \quad \oplus \quad ys \ ; \\
&xs \oplus ys \ \ \cong \ \ ys \oplus xs \ .
\end{aligned}$$

We can prove both bisimilarities in two different ways [14], either by using cofix to build an inhabitant of the coinductive type $max^{\nu_{B\times}\nu_{B\times}}_{B\times\_bisim}$, or by explicitly providing the bisimulation relations and using Lemma 3. In the latter case the two bisimulation relations are given respectively by:

$$\begin{aligned}
&R_1\sigma\tau := \exists x\ xs\ ys,\ \ \sigma = \oplus_3(x, xs, ys) \ \ \wedge \ \ \tau = (\texttt{Cons} \quad x \quad xs) \oplus ys \ \ ; \\
&R_2\sigma\tau := \exists xs\ ys,\ \ \sigma = xs \oplus ys \ \ \wedge \ \ \tau = ys \oplus xs \ \ .
\end{aligned}$$

□

As seen in these examples Theorem 2 provides the bisimilarity equation to recover the specification that was used to forge the parameters T, Λ and g. In general if we want to prove a bisimilarity in *Coq* we have several additional tools:

(i) using the properties of bisimilarities as an equivalence relations and perform equational reasoning;
(ii) using 'type theoretic coinduction', i.e., using cofix and the constructors of coinductive type of $max^{\nu_{B\times}\nu_{B\times}}_{B\times\_bisim}$;

(iii) using conventional coinduction principle and explicitly providing a bisimulation relation between the two sides of the bisimilarity.

We usually apply a combination of the above techniques, but each has characteristics that make it suitable in specific contexts. For example (i) is especially useful when dealing with bisimilarity as a setoid equality, and in combination with other reasoning tools for setoids. Technique (ii) seems to be more suitable for mechanisation as it follows the shape of specifications and leads to smaller *Coq* proof scripts while (iii) is usually more verbose. On the other hand applying (ii) entails that one has to be wary of the guardedness condition as one is using cofix operator of *Coq*, while in (iii) no guardedness check is performed.

As we see λ-coiteration scheme considerably extends the class of functions definable in *Coq*, giving their behavioural equations for free. However, like all syntactic schemes, there is limitation to this scheme, e.g. in [2] it is shown that a specification for the stream of *Hamming* number is not accepted by this scheme.

8 Conclusions and Further Work

We have provided a modular theory of coalgebras in the intensional setting of **CIC** which can be instantiated for specific functors built out of finite sums and coproducts. Each instantiation will give us a theory of bisimilarity which can then be used to build a setoid and work extensionally. Furthermore we showed the usefulness of our coalgebraic setting by developing the λ-coiteration scheme in it and thus extending the class of productive specifications definable in *Coq*. We demonstrated this by an instantiation of our theory for streams and showed some concrete specifications refused by the guardedness condition but accepted using the λ-coiteration scheme in *Coq*.

Our work eases future coalgebraic developments in *Coq*. It is a good evidence that once some technicalities with respect to dependent types are handled most categorical schemes can be translated into intensional type theory. On the other hand it shows that the schemes from category theory can provide suitable workarounds for the restrictions of the guardedness condition without changing the underlying type theory.

The future work would be to build a larger library of results on weakly final coalgebras and developing more powerful definition schemes. Immediate would be the the schemes obtained by adding monadic, pointedness and cofreeness structure on the bi-algebraic nature of λ-coiteration [3]. The long-term challenge would be to extend the formalisation to the powerful functor of Hancock–Setzer [10, 13] and investigating the various schemes there.

References

1. T. Altenkirch, C. McBride, and W. Swierstra. Observational equality, now! In A. Stump and H. Xi, editors, *Proc. of the ACM Workshop Programming Languages meets Program Verification, PLPV 2007, Freiburg, Germany, Oct. 5, 2007*, pages 57–68. ACM Press, 2007.

2. F. Bartels. Generalised coinduction. In A. Corradini, M. Lenisa, and U. Montanari, editors, *Proc. of 4th Workshop on Coalgebraic Methods in Computer Science, CMCS'01, Genova, Italy, 6–7 Apr. 2001*, volume 44(1) of *Electron. Notes Theor. Comput. Sci.*, pages 67–87. Elsevier Science Publishers, 2001.
3. F. Bartels. *On Generalised Coinduction and Probabilistic specification Formats: Distributive laws in coalgebraic modelling*. PhD thesis, Vrije Universiteit Amsterdam, 2004.
4. Y. Bertot and E. Komendantskaya. Inductive and coinductive components of corecursive functions in coq. In J. Adámek and C. Kupke, editors, *Proceedings of the Ninth Workshop on Coalgebraic Methods in Computer Science (CMCS 2008) Budapest, Hungary 4–6 April 2008*, volume 203(5) of *Electron. Notes Theor. Comput. Sci.*, pages 25–47. Elsevier Science Publishers, June 2008.
5. Y. Bertot and E. Komendantskaya. Using structural recursion for corecursion. Technical report, INRIA, Sept. 2008. `http://hal.inria.fr/inria-00322331/en/`, [cited 23 Oct. 2008].
6. The Coq Development Team. *The Coq Proof Assistant Reference Manual, Version 8.1*. LogiCal Project, July 2006. `http://coq.inria.fr/V8.1/refman/index.html`, [cited 23 Oct. 2008].
7. T. Coquand. Infinite objects in type theory. In H. Barendregt and T. Nipkow, editors, *Types for Proofs and Programs, International Workshop TYPES'93, Nijmegen, The Netherlands, May 24–28, 1993, Selected Papers*, volume 806 of *Lecture Notes in Comput. Sci.*, pages 62–78. Springer-Verlag, 1994.
8. P. Di Gianantonio and M. Miculan. A unifying approach to recursive and corecursive definitions. In H. Geuvers and F. Wiedijk, editors, *Types for Proofs and Programs: International Workshop, TYPES 2002, Berg en Dal, The Netherlands, April 24–28, 2002. Selected Papers*, volume 2646 of *Lecture Notes in Comput. Sci.*, pages 148–161. Springer-Verlag, 2003.
9. E. Giménez. *Un Calcul de Constructions Infinies et son Application a la Verification des Systemes Communicants*. PhD thesis PhD 96-11, Laboratoire de l'Informatique du Parallélisme, Ecole Normale Supérieure de Lyon, Dec. 1996.
10. P. Hancock and A. Setzer. Interactive programs and weakly final coalgebras in dependent type theory. In L. Crosilla and P. Schuster, editors, *From Sets and Types to Topology and Analysis. Towards Practicable Foundations for Constructive Mathematics, Proceedings of the workshop, 12–16 May 2003, Venice International University, San Servolo, Venice, Italy.*, volume 48 of *Oxford Logic Guides*, pages 115–134. Oxford University Press, 2005.
11. G. Huet and A. Saïbi. Constructive category theory. In G. D. Plotkin, C. Stirling, and M. Tofte, editors, *Proof, Language, and Interaction, Essays in Honour of Robin Milner*, pages 239–276. The MIT Press, 2000.
12. D. E. Knuth. *The Art of Computer Programming, Seminumerical Algorithms*, volume 2. Addison-Wesley, Reading, Massachusetts, 3rd edition, 1997. xiv+762pp.
13. M. Michelbrink. Interfaces as functors, programs as coalgebras - a final coalgebra theorem in intensional type theory. *Theoret. Comput. Sci.*, 360(1–3):415–439, 2006.
14. M. Niqui. `http://www.cwi.nl/~milad/coalgebras` [cited 23 Oct. 2008], Oct. 2008. Files for Coq v. 8.2beta4.
15. J. J. M. M. Rutten. Universal coalgebra: a theory of systems. *Theoret. Comput. Sci.*, 249(1):3–80, Oct. 2000.
16. J. J. M. M. Rutten. A coinductive calculus of streams. *Math. Structures Comput. Sci.*, 15(1):93–147, 2005.