Terminal semantics for codata types in intensional Martin-Löf type theory

Benedikt Ahrens and Régis Spadotti

Institut de Recherche en Informatique de Toulouse
Université Paul Sabatier, Toulouse

Abstract. We study the notions of *relative comonad* and *comodule over a relative comonad*. We use these notions to give categorical semantics for the coinductive type families of streams and of infinite triangular matrices and their respective cosubstitution operations in intensional Martin-Löf type theory. Our results are mechanized in the proof assistant `Coq`.

1 Introduction

In this work, we study the notions of *relative comonad* and *comodule over a relative comonad*. We then use these notions to characterize two coinductive data types and their respective cosubstitution operations in intensional Martin-Löf type theory via a universal property: firstly, we consider the *homogeneous* codata type family of streams parametrized by a base type. Secondly, we consider the *heterogeneous* codata type family of infinite triangular matrices parametrized by the type of diagonal entries. In the rest of the introduction, we explain some of the vocabulary occurring in these first sentences.

1.1 Semantics of inductive types

Before turning to *co*inductive types, we give an overview over semantics for inductive types.

In a set-theoretic setting, inductive sets are characterized as initial algebras for some endofunctor on the category of sets [15].

In a type-theoretic setting as given by Martin-Löf type theory [17], two approaches to the semantics of inductive types have been studied: one approach consists in showing that inductive types exist in a *model* of the type theory, as is done, e.g., by Moerdijk and Palmgren [19]. Another approach is to prove that adding certain type-theoretic rules to the type theory—rules which postulate the existence of inductive types in the type theory—implies (or is equivalent to) the existence of a universal object *within* type theory (see, e.g., [8, 11]). This latter approach is the one we adopt in the present work: we add axioms to intensional Martin-Löf type theory, postulating the existence of *co*inductive types, see below.

Some attention has to be given to the precise formulation of the type theory in question, in particular, to the nature of *equality* in the type theory. There are two notions of equality in Martin-Löf type theory, an external (judgmental) one, and an internal (propositional) one. The latter is internalized by the *identity type*, a type family that associates to any two inhabitants $a, b : A$ of a type A the type of (propositional) "identities" between them. Judgmentally equal terms are always propositionally equal, but the converse is not always true. One distinguishes two variants of type theory: in *extensional* type theory, propositional equalities (i.e. terms of identity type) are reflected, via a *reflection rule*, into the judgmental equality of the type theory. Here, judgmental and propositional equality thus coincide. In *intensional* type theory, there is no such reflection rule.

The reflection rule equips *extensional* type theory with extensional features similar to those of set theory. As a consequence, the characterization given by Dybjer [11] of an inductive type in extensional MLTT as initial algebra for some endofunctor on the category of types works as in the category of sets.

Intensional Martin-Löf type theory [17] lacks the reflection principle for the sake of decidability of type checking. It forms the base of two computer proof assistants, `Coq` and `Agda`. The characterization of inductive types as initial algebras of some endofunctor in intensional Martin-Löf type theory fails [11], but it can be recovered [8] in an *extension* of intensional MLTT called *Homotopy Type Theory* [22]. In this extension, function extensionality is provable from the *Univalence Axiom*. For a suitable definition of *uniqueness*—*contractibility* in HoTT jargon—one can prove uniqueness of the algebra morphisms out of the one whose carrier is given by the inductive type under consideration. The mentioned work [8] thus shows that the characterization of inductive types as initial algebras carries over from extensional to intensional type theory if one adds an extensionality principle for functions and adapts the notion of uniqueness.

The characterization of inductive sets/types as initial objects in some category has been extended to some *heterogeneous*—also called *nested*—inductive data types, e.g., the type of λ-terms, in different formulations [12, 13]. The main goal of these works is not just to characterize a data type via a universal property, but rather a data type *equipped with a canonical, well-behaved substitution operation.*

1.2 Coinductive types: simply dualizing?

In a traditional set-theoretic setting, the theory of coinductive sets is completely dual to that of inductive sets: dually to inductive sets, *co*inductive sets such as streams are characterized as terminal coalgebras of suitable endofunctors [15].

In type theory, this duality between inductive and coinductive objects breaks. This is rooted in the unsuitability of the Martin-Löf identity type to express sameness for inhabitants of coinductive types: while identity terms are inductively generated and hence necessarily finite, a proof of sameness between coinductive terms—which represent infinite objects—constitutes a potentially infinite object itself. The type of such proofs hence cannot be exhaustively given by the (inductive) identity type.

Instead of comparing two coinductive terms in Martin-Löf type theory up to identity, one defines a binary coinductive relation called *bisimilarity* on a given coinductive type, with respect to which one compares its inhabitants [10].

Categorically, *bisimilarity* is given as the greatest—*weakly terminal—relation* on the coinductive type that is compatible with the coalgebra structure.

Consequently, we consider two maps into a coinductive type to be the same if they are *pointwise bisimilar*—an analogue to the aforementioned principle of function extensionality available in Homotopy Type Theory. With these conventions, we give, in the present work, a characterization of some coinductive data types as *terminal* object in some category defined in intensional Martin-Löf type theory. More precisely, we consider an example of *homogeneous* codata type, streams, and an example of *heterogeneous* codata type, triangular matrices. For each of these examples we prove, from type-theoretic rules specifying the respective codata type added to the basic rules of Martin-Löf type theory, the existence of a terminal object in some category *within IMLTT*. The objects of the considered categories are not plain coalgebras for some endofunctor, but rather coalgebras with extra structure. This extra structure accounts for a *cosubstitution* operation with which one can equip the considered codata types *in a canonical way*. The cosubstitution operation is explained in the next section.

We mention here work by Marchi [16], who studies the existence of coinductive types in models of *extensional* Martin-Löf type theory. In that work, no *internal* characterization of coinductive types is given. Furthermore, his work does not account for the mentioned cosubstitution operation.

1.3 Cosubstitution is a comonadic operation on parametrized codata.

Many inductive types, in particular parametrized ones representing *syntax* such as the lambda calculus, are canonically equipped with a substitution operation. The categorical characterization of this substitution operation has been studied extensively; our list of pointers [7, 12, 20, 21, 13, 6, 4] is necessarily incomplete. One such characterization is via *monads*; substitution can be seen as part of a monadic structure on the functor given by the parametrized data type [7, 13, 6, 4].

Dually, coinductive types can often be equipped with a *cosubstitution* operation. Is this cosubstitution operation part of a comonad structure? In a set-theoretic setting, the fact that cosubstitution for coinductive sets is comonadic is established by Uustalu and Vene [23].

In IMLTT however, in order to characterize that cosubstitution operation on a given codata type, and its algebraic properties, we develop the notion of *relative comonad* and *comodule over a relative comonad*. The need to consider *relative* comonads arises from the need to check the algebraic properties of cosubstitution modulo *bisimilarity* rather than modulo (the) identity (type).

In order to integrate a notion of cosubstitution into the categorical semantics of the coinductive types under consideration, we do not consider a category of plain coalgebras for some endofunctor. Instead, we consider coalgebras with extra structure accounting for a comonadic operation, which for the terminal coalgebra is given by cosubstitution. Our terminal semantics characterizes not only the codata types themselves but also the bisimilarity relation and—via the use of (relative) comonads—a canonical cosubstitution operation on them.

1.4 Our proofs are correct.

All our results have been implemented in the proof assistant `Coq` [9]. The `Coq` source files and HTML documentation are available online [5]. Here, we hence omit the proofs and focus on definitions and statements.

Disclaimer The category-theoretic concepts studied in this work are agnostic to the foundational system being worked in. While we present them in a type-theoretic style, the definitions and lemmas can trivially be transferred to a set-theoretic setting. Throughout this article, we use type-theoretic notation, writing $t : T$ to indicate that t is of type T. For instance, we write $f : \mathcal{C}(A, B)$ to indicate that f is a morphism from object A to object B in category $\mathcal{C}$. Whenever an operation takes several arguments, we write some of them as indices; these indices might be omitted when they can be deduced from the type of the later arguments. We assume basic knowledge of category theory; any instances used are defined in the following.

Organisation of the paper In Section 2 we introduce some concepts and notations used later on. In Section 3 we present the coinductive type families Stream of streams and Tri of infinite triangular matrices and some operations on those codata types. Their specifying rules are given in Figure 1 and Figure 3, respectively. In Section 4 we present *relative comonads* and define the category of comonads relative to a fixed functor. We give some examples of such structures, using the codata types presented in Section 3. In Section 5 we define comodules over relative comonads and give some constructions of comodules. Again, examples of such structures are taken from Section 3. In Section 6 we define categories of models for the codata types presented in Section 3, based on the category-theoretic notions developed in the previous sections. We then prove that the codata types constitute the terminal model in the respective categories. Finally, we present an example of a map defined as a terminal map exploiting the universal property of streams. In Section 7 we explain some details of the formalization of this work in the proof assistant `Coq`. A table with the correspondence between formal and informal definitions is given in Figure 5.

2 Preliminaries

We present some particular categories used later on, and fix some notation.

Definition 1. *A* setoid *in intensional Martin-Löf type theory is a pair* $(A, \sim_A)$ *of a type* A *together with an equivalence relation* $\sim_A$ *on* A*. Given a setoid* $S = (A, \sim_A)$*, we often abuse notation by writing* $s : S$ *instead of* $s : A$ *for* *a term s of A. Given two setoids $(A, \sim_A)$ and $(B, \sim_B)$, the cartesian product $A \times B$ of their underlying types is equipped with an equivalence relation given component-wise, thus yielding a product on setoids.*

Definition 2. *A* category $\mathcal{C}$ *in intensional Martin-Löf type theory is given by*

- *a type of objects, also denoted by $\mathcal{C}$;*
- *for any two objects $a, b : \mathcal{C}$, a* setoid $\mathcal{C}(a, b)$ *of morphisms from a to b;*
- *an identity morphism $\mathsf{id}_a : \mathcal{C}(a, a)$ for any $a : \mathcal{C}$;*
- *a dependent composition operation $(\ \circ\ )_{a,b,c} : \mathcal{C}(b, c) \times \mathcal{C}(a, b) \to \mathcal{C}(a, c)$;*
- *a proof that composition is compatible with the equivalence relations of the setoids of morphisms;*
- *proofs of unitality and associativity of composition stated in terms of the equivalence relations on the morphisms.*

This definition has been used from early on, e.g., in [3, 14]). We write $f : a \to b$ for a morphism $f : \mathcal{C}(a, b)$, when the category $\mathcal{C}$ can be deduced from the context.

Functors and natural transformations are defined in terms of the setoidal equivalence relations on morphisms. We omit the definitions, which can be found in our `Coq` files [5].

Definition 3. *We denote by* Type *the category (in IMLTT) of types (of a fixed universe) and total functions between them in Martin-Löf type theory. The setoidal equivalence relation on* $\mathsf{Type}(A, B)$ *is given by pointwise equality as given by the Martin-Löf identity type,* $f \sim g \quad =_{def} \quad \forall x : A, fx = gx$.

We denote by Setoid *the category an object of which is a setoid. A morphism between setoids is a type-theoretic function between the underlying types that is compatible in the obvious sense with the equivalence relations of the source and target setoids. The category* Setoid *is cartesian closed and hence a category in the sense of Definition 2: two parallel morphisms of setoids $f, g : A \to B$ are equivalent if for any $a : A$ we have $fa \sim_B ga$.*

Definition 4. *The functor* $\mathsf{eq} : \mathsf{Type} \to \mathsf{Setoid}$ *is defined as the left adjoint to the forgetful functor* $U : \mathsf{Setoid} \to \mathsf{Type}$*. Explicitly, the functor* eq *sends any type X to the setoid $(X, =_X)$ given by the type X itself, equipped with the propositional equality relation $=_X$ specified via Martin-Löf's identity type on X.*

Remark 5 (Notation for product). We denote the category-theoretic binary product of objects A and B of a category $\mathcal{C}$ by $A \times B$. We write $\mathsf{pr}_1(A, B) : \mathcal{C}(A \times B, A)$ and $\mathsf{pr}_2(A, B) : \mathcal{C}(A \times B, B)$ for the projections, occasionally omitting the argument (A, B). Given $f : \mathcal{C}(A, B)$ and $g : \mathcal{C}(A, C)$, we write $\langle f, g \rangle : \mathcal{C}(A, B \times C)$ for the induced map into the product such that $\mathsf{pr}_1 \circ \langle f, g \rangle = f$ and $\mathsf{pr}_2 \circ \langle f, g \rangle = g$.

Both of the categories of Definition 3 have finite products.

Definition 6. *A functor $F : \mathcal{C} \to \mathcal{D}$ between categories with finite products* preserves products *if, for any two objects A and B of $\mathcal{C}$, the morphism $\phi^F_{A,B}$ is an isomorphism, where*

$$\phi^F_{A,B} := \big\langle F(\mathsf{pr}_1), F(\mathsf{pr}_2) \big\rangle : \mathcal{D}\big(F(A \times B), FA \times FB\big) \ .$$

Example 7. The functor $\mathsf{eq} : \mathsf{Type} \to \mathsf{Setoid}$ of Definition 4 preserves products.

3 Codata types in intensional Martin-Löf type theory

We consider two particular coinductive type families in Intensional Martin-Löf type theory (IMLTT) [17], a type-theoretic foundational system. For $a, b : A$, we denote by $a = b$ the Martin-Löf identity type between a and b.

In this section, we present the type-theoretic rules for these codata types and *bisimilarity* on them. Categorically, bisimilarity on a given codata type is given by the largest binary relation on that type that is compatible with the destructors (elimination); it is the appropriate notion of sameness on inhabitants of these types [10]. A coinductive type with bisimilarity forms a setoid as in Definition 3. We thus denote bisimilar elements using an infix $\sim$, as in $t \sim t'$.

A map into a codata type is defined using its *introduction* rule. Intuitively, the introduction rule takes as arguments the *observations* one can make on the image of the map thus defined. For instance, a map into streams is defined by specifying head and tail on the output of that map: this intuition can be obtained by considering the combination of introduction and computation rules from Figure 1. We thus use a kind of "copattern matching" [2] to define maps, as follows: the clauses $\mathsf{head} \circ f := h$ and $\mathsf{tail} \circ f := t$ abbreviate the definition $f := \mathsf{corec}(h, t)$.

The first example is the type of *streams* of elements of a given base type A:

Example 8. Let A be a type. The type—setoid, actually—$\mathsf{Stream}A$ of *streams over* A is coinductively defined via the destructors given in Figure 1.

We define a *co*substition operation

$$\mathsf{cosubst}_{A,B} : (\mathsf{Stream}A \to B) \to \mathsf{Stream}A \to \mathsf{Stream}B$$

on streams via the following clauses:

$$\mathsf{head} \circ \mathsf{cosubst}\ f := f \quad \text{and} \quad \mathsf{tail} \circ \mathsf{cosubst}\ f := \mathsf{cosubst}\ f \circ \mathsf{tail} \ . \tag{3.1}$$

We call such an operation "cosubstitution" since its type is dual to, e.g., the simultaneous substitution operation of the lambda calculus [7]. Cosubstitution is compatible with bisimilarity on streams, and thus is (the carrier of) a family

$$\mathsf{cosubst}_{A,B} : \mathsf{Setoid}(\mathsf{Stream}A, \mathsf{eq}B) \to \mathsf{Setoid}(\mathsf{Stream}A, \mathsf{Stream}B) \ .$$

Streams are node-labeled trees where every node has exactly one subtree. We also consider a type of trees where every node has an arbitrary, but fixed, number of subtrees, parametrized by a type B.

Example 9 (Node-labeled trees). We denote by $\mathsf{Tree}_B(A)$ the codata type given by one destructor head and a family of destructors $(\mathsf{tail}_b)_{b:B}$ with types analogous to those defining Stream of Example 8. We thus obtain Stream by considering, for B, the singleton type.

We furthermore consider the *heterogeneous* codata type family of *infinite triangular matrices*:

Rules for Stream

Formation

$$\frac{A : \mathsf{Type}}{\mathsf{Stream} A : \mathsf{Type}}$$

Elimination

$$\frac{t : \mathsf{Stream} A}{\mathsf{head}_A\ t : A} \qquad \frac{t : \mathsf{Stream} A}{\mathsf{tail}_A\ t : \mathsf{Stream} A}$$

Introduction

$$\frac{T : \mathsf{Type} \qquad hd : T \to A \qquad tl : T \to T}{\mathsf{corec}_A\ hd\ tl : T \to \mathsf{Stream} A}$$

Computation

$$\frac{hd : T \to A \qquad tl : T \to T \qquad t : T}{\mathsf{head}_A(\mathsf{corec}_A\ hd\ tl\ t) = hd(t)}$$

$$\frac{hd : T \to A \qquad tl : T \to T \qquad t : T}{\mathsf{tail}_A(\mathsf{corec}_A\ hd\ tl\ t) = \mathsf{corec}_A\ hd\ tl\ (tl\ t)}$$

Bisimilarity on Stream

Formation

$$\frac{A : \mathsf{Type} \qquad s, t : \mathsf{Stream} A}{\mathsf{bisim}_A\ s\ t : \mathsf{Type}}$$

Elimination

$$\frac{s, t : \mathsf{Stream} A \qquad p : \mathsf{bisim}_A\ s\ t}{\mathsf{head}_A\ s = \mathsf{head}_A\ t} \qquad \frac{s, t : \mathsf{Stream} A \qquad p : \mathsf{bisim}_A\ s\ t}{\mathsf{bisim}_A(\mathsf{tail}_A\ s)(\mathsf{tail}_A\ t)}$$

Introduction

$$\frac{\begin{array}{c} R : \mathsf{Stream} A \to \mathsf{Stream} A \to \mathsf{Type} \\ \forall\ s, t : \mathsf{Stream} A, R\ s\ t \to \mathsf{head}\ s = \mathsf{head}\ t \\ \forall\ s, t : \mathsf{Stream} A, R\ s\ t \to R\ (\mathsf{tail}\ s)(\mathsf{tail}\ t) \end{array}}{\forall\ s, t : \mathsf{Stream} A, R\ s\ t \to \mathsf{bisim}\ s\ t}$$

Fig. 1. Rules for streams and bisimilarity on them

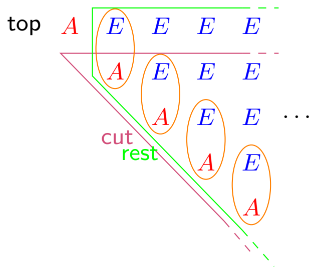

Fig. 2. An infinite triangular matrix over type A and various operations

Example 10. Infinite triangular matrices are studied in detail by Matthes and Picard [18], and also by Abel, Matthes, and Uustalu [1]. We give a brief summary, but urge the reader to consult the given reference for an in-depth explanation. The codata type family Tri of infinite triangular matrices is parametrized by a fixed type E for entries not on the diagonal, and indexed by another, *variable*, type A for entries on the diagonal. Schematically, such a matrix looks like in Figure 2. The codata type is specified via two destructors top and rest, whose types are given in Figure 3. Given a matrix over type A, its rest—obtained by removing the first element on the diagonal, i.e. the top element—can be considered as a trapezium as indicated by the green line in Figure 2, or alternatively, as a triangular matrix over type $E \times A$, by bundling the entries of the diagonal with those above as indicated by the orange frames in Figure 2. The latter representation is reflected in the type of the destructor rest. This change of parameter (from A to $E \times A$) in the type of rest is why the family Tri is called "heterogeneous".

A cosubstitution operation, "redecoration", $\mathsf{redec}_{A,B} : (\mathsf{Tri}A \to B) \to \mathsf{Tri}A \to \mathsf{Tri}B$ is defined through the clauses

$$\mathsf{top} \circ \mathsf{redec}\ f := f \qquad \text{and} \qquad \mathsf{rest} \circ \mathsf{redec}\ f := \mathsf{redec}\ (\mathsf{extend}\ f) \circ \mathsf{rest}\ . \tag{3.2}$$

Here, the family of functions $\mathsf{extend}_{A,B} : (\mathsf{Tri}A \to B) \to \mathsf{Tri}(E \times A) \to E \times B$ is suitably defined to account for the change of the type of the argument of redec when redecorating $\mathsf{rest}\ t : \mathsf{Tri}(E \times A)$ rather than $t : \mathsf{Tri}A$, namely

$$\mathsf{extend}(f) := \langle \mathsf{pr}_1(E, A) \circ \mathsf{top}_{E\times A}, f \circ \mathsf{cut}_A \rangle\ .$$

The auxiliary function $\mathsf{cut}_A : \mathsf{Tri}(E \times A) \to \mathsf{Tri}A$ is defined corecursively via

$$\mathsf{top} \circ \mathsf{cut} := \mathsf{pr}_2 \circ \mathsf{top} \qquad \text{and} \qquad \mathsf{rest} \circ \mathsf{cut} := \mathsf{cut} \circ \mathsf{rest}\ .$$

All the operations are suitably compatible with the bisimilarity relations, so that they can be equipped with the types

$$\begin{aligned} \mathsf{redec}_{A,B} &: \mathsf{Setoid}(\mathsf{Tri}A, \mathsf{eq}B) \to \mathsf{Setoid}(\mathsf{Tri}A, \mathsf{Tri}B) \\ \mathsf{extend}_{A,B} &: \mathsf{Setoid}(\mathsf{Tri}A, \mathsf{eq}B) \to \mathsf{Setoid}\big(\mathsf{Tri}(E \times A), \mathsf{eq}(E \times B)\big) \\ \mathsf{cut}_A &: \mathsf{Setoid}(\mathsf{Tri}(E \times A), \mathsf{Tri}A)\ . \end{aligned}$$

Rules for Tri
Formation

$$\frac{A : \mathsf{Type}}{\mathsf{Tri}A : \mathsf{Type}}$$

Elimination

$$\frac{t : \mathsf{Tri}A}{\mathsf{top}_A\ t : A} \qquad \frac{t : \mathsf{Tri}A}{\mathsf{rest}_A\ t : \mathsf{Tri}(E \times A)}$$

Introduction

$$\frac{T : \mathsf{Type} \to \mathsf{Type} \qquad hd : \forall A, TA \to A \qquad tl : \forall A, TA \to T(E \times A)}{\mathsf{corec}_T\ hd\ tl : \forall A, TA \to \mathsf{Tri}A}$$

Computation

$$\frac{hd : \forall A, TA \to A \qquad tl : \forall A, TA \to T(E \times A) \qquad t : TA}{\mathsf{top}_T(\mathsf{corec}_A\ hd\ tl\ t) = hd(t)}$$

$$\frac{hd : \forall A, TA \to A \qquad tl : \forall A, TA \to T(E \times A) \qquad t : TA}{\mathsf{rest}_T(\mathsf{corec}_A\ hd\ tl\ t) = \mathsf{corec}_A\ hd\ tl\ (tl\ t)}$$

Bisimilarity for Tri

Formation

$$\frac{A : \mathsf{Type} \qquad s, t : \mathsf{Tri}A}{\mathsf{bisim}_A\ s\ t : \mathsf{Type}}$$

Elimination

$$\frac{s, t : \mathsf{Tri}A \qquad p : \mathsf{bisim}_A\ s\ t}{\mathsf{top}_A\ s = \mathsf{top}_A\ t} \qquad \frac{s, t : \mathsf{Tri}A \qquad p : \mathsf{bisim}_A\ s\ t}{\mathsf{bisim}_A(\mathsf{rest}_A\ s)(\mathsf{rest}_A\ t)}$$

Introduction

$$\frac{\begin{array}{c} R : \forall A, \mathsf{Tri}A \to \mathsf{Tri}A \to \mathsf{Type} \\ \forall A, \forall\ s, t : \mathsf{Tri}A, R\ s\ t \to \mathsf{top}\ s = \mathsf{top}\ t \\ \forall A, \forall\ s, t : \mathsf{Tri}A, R\ s\ t \to R\ (\mathsf{tail}\ s)(\mathsf{tail}\ t) \end{array}}{\forall A, \forall\ s, t : \mathsf{Tri}A, R\ s\ t \to \mathsf{bisim}\ s\ t}$$

Fig. 3. Rules for triangular matrices and bisimilarity on them

Note how heterogeneity of the destructor rest makes the definition of redec considerably more complicated than that of the analogous operation $\mathsf{cosubst}$ on streams.

Remark 11. We do not specify any computation rules for bisimilarity, i.e., we think of bisimilarity as a proof-irrelevant notion. Categorically, the lack of such a computation rule amounts to saying that bisimilarity is a *weakly* terminal bisimulation on the respective codata. (Here, a bisimulation is a binary relation on the codata type that is compatible with the destructors of the codata types.) In the setting of Homotopy Type Theory [22], where one would identify bisimilarity with propositional equality, a proof-relevant treatment would be adequate.

4 Relative comonads and their morphisms

In this section we define the category of *comonads relative to a fixed functor*, and present some examples of such comonads and their morphisms.

Relative monads were defined by Altenkirch, Chapman, and Uustalu [6] as a notion of monad-like structure whose underlying functor is not necessarily an endofunctor. The dual notion is that of a relative *co*monad:

Definition 12. *Let $F : \mathcal{C} \to \mathcal{D}$ be a functor. A* ***relative comonad*** T ***over*** F *is given by*

- *a map $T : \mathcal{C}_0 \to \mathcal{D}_0$ on the objects of the categories involved;*
- *an operation* $\mathsf{counit} : \forall A : \mathcal{C}_0, \mathcal{D}(TA, FA)$*;*
- *an operation* $\mathsf{cobind} : \forall A, B : \mathcal{C}_0, \mathcal{D}(TA, FB) \to \mathcal{D}(TA, TB)$ *such that*
- $\forall A, B : \mathcal{C}_0, \forall f : \mathcal{D}(TA, FB), \mathsf{counit}_B \circ \mathsf{cobind}(f) = f$*;*
- $\forall A : \mathcal{C}_0, \mathsf{cobind}(\mathsf{counit}_A) = \mathsf{id}_{TA}$*;*
- $\forall A, B, C : \mathcal{C}_0, \forall f : \mathcal{D}(TA, FB), \forall g : \mathcal{D}(TB, FC),$
  $\mathsf{cobind}(g) \circ \mathsf{cobind}(f) = \mathsf{cobind}(g \circ \mathsf{cobind}(f))$.

Just like relative monads, relative comonads are functorial:

Definition 13. *Let T be a comonad relative to $F : \mathcal{C} \to \mathcal{D}$. For $f : \mathcal{C}(A, B)$ we define* $\mathsf{lift}^T(f) := \mathsf{cobind}(Ff \circ \mathsf{counit}_A) : \mathcal{D}(TA, TB)$*. The functor properties are easily checked.*

Remark 14. It follows from the comonadic axioms that counit and cobind are natural transformations with respect to the functoriality defined in Definition 13:

$$\begin{aligned}
\mathsf{counit} &: T \to F : \mathcal{C} \to \mathcal{D} \\
\mathsf{cobind} &: \mathcal{D}(T\_, F\_) \to \mathcal{D}(T\_, T\_) : \mathcal{C}^{\mathrm{op}} \times \mathcal{C} \to \mathsf{Set} \quad .
\end{aligned}$$

Relative comonads over the identity functor are exactly comonads.

Example 15 (Relative comonads from comonads). Let $F : \mathcal{C} \to \mathcal{D}$ be a fully faithful functor and $(M, \mathsf{counit}, \mathsf{cobind})$ be a (traditional) comonad (in Kleisli form) on $\mathcal{C}$. We define a comonad FM relative to F by setting:

- $FM(A) := F(MA)$;
- $\mathsf{counit}^{FM}_A := F(\mathsf{counit}^M_A) : \mathcal{D}(FMA, FA)$;
- $\mathsf{cobind}^{FM}_{A,B}(f) := F\big(\mathsf{cobind}^M_{A,B}(F^{-1}f)\big)$.

The proof of the axioms of a relative comonad is immediate.

Example 16 (Relative comonads from comonads II). Let $F : \mathcal{C} \to \mathcal{D}$ be a functor and $(M, \mathsf{counit}, \mathsf{cobind})$ be a (traditional) comonad (in Kleisli form) on $\mathcal{D}$. We define a comonad MF relative to F by setting:

- $MF(A) := M(FA)$;
- $\mathsf{counit}^{MF}_A := \mathsf{counit}^M_{FA} : \mathcal{D}(MFA, FA)$;
- $\mathsf{cobind}^{MF}_{A,B}(f) := \mathsf{cobind}^M_{FA,FB}(f) : \mathcal{D}(MFA, MFB)$ for $f : \mathcal{D}(MFA, FB)$.

The proof of the axioms of a relative comonad is immediate.

Example 17 (Streams). The codata type family $\mathsf{Stream} : \mathsf{Type} \to \mathsf{Setoid}$ of Example 8 is equipped with a structure of a comonad relative to the functor $\mathsf{eq} : \mathsf{Type} \to \mathsf{Setoid}$ with $\mathsf{counit}_A := \mathsf{head}_A$ and $\mathsf{cobind}_{A,B} := \mathsf{cosubst}_{A,B}$.

Remark 18. Instead of considering Stream as a relative comonad from Type to Setoid, it could be considered as a traditional comonad on the category Setoid, propagating the equivalence relation on the base type, say A, to $\mathsf{Stream}A$. The relative point of view can then be recovered as an instance of Example 16.

Example 19 (Trees). Fix a type B. Analogously to Example 17, the map $A \mapsto \mathsf{Tree}_B(A)$ of Example 9 is equipped with a structure of a comonad relative to $\mathsf{eq} : \mathsf{Type} \to \mathsf{Setoid}$.

Example 20 (Infinite triangular matrices). The codata type family $\mathsf{Tri} : \mathsf{Type} \to \mathsf{Setoid}$ of Example 10 is equipped with a structure of a comonad relative to the functor $\mathsf{eq} : \mathsf{Type} \to \mathsf{Setoid}$ with $\mathsf{counit}_A := \mathsf{top}_A$ and $\mathsf{cobind}_{A,B} := \mathsf{redec}_{A,B}$.

Remark 21. A *weak constructive comonad* as defined by Matthes and Picard [18] to characterize the codata type Tri and redecoration on it, is *precisely* a comonad relative to the functor $\mathsf{eq} : \mathsf{Type} \to \mathsf{Setoid}$.

The notion of relative comonad captures many properties of Stream resp. Tri and cosubstitution on them, in particular the interplay of cosubstitution with the destructors head resp. top via the first two axioms. In order to capture the interplay of cosubstitution with the destructor tail resp. rest, we develop the notion of *comodule over a relative comonad* in Section 5.

Morphisms of relative comonads are families of morphisms that are compatible with the comonadic structure:

Definition 22. *Let T and S be comonads relative to a functor $F : \mathcal{C} \to \mathcal{D}$. A* ***morphism of relative comonads*** *$\tau : T \to S$ is given by a family of morphisms $\tau_A : \mathcal{D}(TA, SA)$ such that for any $A : \mathcal{C}_0$, $\mathsf{counit}^T_A = \mathsf{counit}^S_A \circ \tau_A$ and for any $A, B : \mathcal{C}_0$ and $f : \mathcal{D}(SA, FB)$, $\tau_B \circ \mathsf{cobind}^T(f \circ \tau_A) = \mathsf{cobind}^S(f) \circ \tau_A$.*

Relative comonads over a fixed functor F and their morphisms form a category $\mathsf{RComon}(F)$ with the obvious identity and composition operations.

Remark 23. A morphism $\tau : T \to S$ of relative comonads over a functor $F : \mathcal{C} \to \mathcal{D}$ is *natural* with respect to the functorial action of Definition 13.

Example 24. Continuing Example 15 with M, M' two monads on $\mathcal{C}$, given a comonad morphism $\tau : M \to M'$, one obtains a morphism of relative comonads $F\tau : FM \to FM'$ by setting $F\tau_A := F(\tau_A)$. Again, the axioms are easy to check.

Remark 25. The definitions given in Example 15 and Example 24 yield a functor from comonads on $\mathcal{C}$ to comonads relative to $F : \mathcal{C} \to \mathcal{D}$. If F is a right adjoint with left adjoint L, $L \dashv F$, then postcomposing a comonad T relative to F with the functor L yields a monad on $\mathcal{C}$. Again, this map extends to morphisms. The two functors between categories of monads thus defined are again adjoints. Writing down the details is lengthy but easy.

For instance, in a type theory with quotients, such as the *Univalent Foundations* a.k.a. *Homotopy Type Theory* [22], the functor "quotient" from setoids to types is left adjoint to the fully faithful functor $\mathsf{eq} : \mathsf{Type} \to \mathsf{Setoid}$, thus above construction is applicable.

Example 26. We define a morphism of relative comonads $\mathsf{diag} : \mathsf{Tri} \to \mathsf{Stream}$ as follows. Given a matrix $t : \mathsf{Tri}A$, its diagonal is a stream $\mathsf{diag}_A\ t : \mathsf{Stream}A$. The map diag_A is defined via the clauses

$$\mathsf{head} \circ \mathsf{diag}_A := \mathsf{top} \quad \text{and} \quad \mathsf{tail} \circ \mathsf{diag}_A := \mathsf{diag}_A \circ \mathsf{cut} \circ \mathsf{rest} \ .$$

Remark 27. The destructors tail (for Stream) and rest (for Tri) are *not* comonad morphisms. One can, however, equip the functor given by precomposing Tri with "product with E", i.e. $A \mapsto \mathsf{Tri}(E \times A)$, with a structure of relative comonad, induced by that on Tri, cf. Definition 28.

Definition 28. *Let T be a comonad relative to a product-preserving functor $F : \mathcal{C} \to \mathcal{D}$ between categories with finite products, and let $E : \mathcal{C}_0$ be a fixed object of $\mathcal{C}$. The map $A \mapsto T(E \times A)$ inherits the structure of a comonad relative to F from T: the counit is defined as*

$$\mathsf{counit}_A := \mathsf{counit}^T_A \circ \mathsf{lift}^T(\mathsf{pr}_2(E, A))$$

and the cobind operation as

$$\begin{aligned} \mathsf{cobind}_{A,B} : \mathcal{D}\big(T(E \times A), FB\big) &\to \mathcal{D}\big(T(E \times A), T(E \times B)\big) \\ f &\mapsto \mathsf{cobind}^T(\mathsf{extend}'\ f) \end{aligned}$$

with extend' *defined as*

$$\begin{aligned} \mathsf{extend}' : \mathcal{D}\big(T(E \times A), FB\big) &\to \mathcal{D}\big(T(E \times A), F(E \times B)\big) \ , \\ f &\mapsto {\phi^F_{E,B}}^{-1} \circ \langle \mathsf{counit}^T_E \circ T(\mathsf{pr}_1), f \rangle \ . \end{aligned}$$

Remark 29 (Comonads into cartesian closed categories). With the notations of Definition 12, suppose that the category $\mathcal{D}$ is cartesian closed, i.e. that $\mathcal{D}$ has internal hom-objects. This is the case, e.g., for the category Setoid of setoids. The cobind operation of a relative comonad into $\mathcal{D}$ might then be defined to be a family of arrows in $\mathcal{D}$,

$$\mathsf{cobind}_{A,B} : \underline{\mathcal{D}}(TA, FB) \to \underline{\mathcal{D}}(TA, TB) \ . \tag{4.1}$$

The comonads Stream (cf. Example 17) and Tri (cf. Example 19) are comonads with such a cobind operation. In these two cases, this "enriched" definition of the cobind operation given in Equation (4.1) encodes a higher-order compatibility of cosubstitution with bisimilarity, namely that cosubstitution with two extensionally bisimilar functions yields extensionally bisimilar functions.

5 Comodules over relative comonads

In this section we develop the notion of *comodule over a relative comonad*, dualizing the notion of module over a relative monad [4].

Definition 30. *Let T be a comonad relative to $F : \mathcal{C} \to \mathcal{D}$, and let $\mathcal{E}$ be a category. A* ***comodule over T towards*** *$\mathcal{E}$ consists of*

- *a map $M : \mathcal{C}_0 \to \mathcal{E}_0$ on the objects of the categories involved and*
- *an operation* $\mathsf{mcobind} : \forall A, B : \mathcal{C}_0, \mathcal{D}(TA, FB) \to \mathcal{E}(MA, MB)$ *such that*
- $\forall A : \mathcal{C}_0, \mathsf{mcobind}(\mathsf{counit}_A) = \mathsf{id}_{MA}$*;*
- $\forall A, B, C : \mathcal{C}_0, \forall f : \mathcal{D}(TA, FB), \forall g : \mathcal{D}(TB, FC),$
  $\mathsf{mcobind}(g) \circ \mathsf{mcobind}(f) = \mathsf{mcobind}(g \circ \mathsf{cobind}(f))$.

Every relative comonad comes with a canonical comodule over itself:

Definition 31. *Given a comonad T relative to $F : \mathcal{C} \to \mathcal{D}$, the map $A \mapsto TA$ yields a comodule over T with target category $\mathcal{D}$, the* ***tautological comodule*** *of T, also called T. The comodule operation is given by* $\mathsf{mcobind}^T(f) := \mathsf{cobind}^T(f)$.

Similarly to relative comonads, comodules over these are functorial:

Definition 32. *Let M : $\mathsf{RComod}(T, \mathcal{E})$ be a comodule over T towards some category $\mathcal{E}$. For $f : \mathcal{C}(A, B)$ we define*

$$\mathsf{mlift}^M(f) := \mathsf{mcobind}(Ff \circ \mathsf{counit}_A).$$

A more interesting example of comodule is given by the functor that maps a type A to the setoid $\mathsf{Tri}(E \times A)$ for some fixed type E:

Example 33. The map $A \mapsto \mathsf{Tri}(E \times A)$ is equipped with a comodule structure over the relative comonad Tri by defining the comodule operation $\mathsf{mcobind}$ as (cf. Example 10) $\mathsf{mcobind}_{A,B}(f) := \mathsf{redec}(\mathsf{extend}\ f)$.

A *morphism of comodules* is given by a family of morphisms that is compatible with the comodule operation:

Definition 34. *Let $M, N : \mathcal{C} \to \mathcal{E}$ be comodules over the comonad T relative to $F : \mathcal{C} \to \mathcal{D}$. A **morphism of comodules** from M to N is given by a family of morphisms $\alpha_A : \mathcal{E}(MA, NA)$ such that for any $A, B : \mathcal{C}_0$ and $f : \mathcal{D}(TA, FB)$ one has $\alpha_B \circ \mathsf{mcobind}^M(f) = \mathsf{mcobind}^N(f) \circ \alpha_A$.*

Example 35. The destructor $\mathsf{tail}_A : \mathsf{Stream}A \to \mathsf{Stream}A$ is the carrier of a morphism of tautological comodules (over the relative comonad Stream).

Example 36. The destructor rest of Example 10 is a morphism of comodules over the comonad Tri from the tautological comodule Tri to the comodule $\mathsf{Tri}(E \times \_)$.

Composition and identity of comodule morphisms happens pointwise. We thus obtain a category $\mathsf{RComod}(T, \mathcal{E})$ of comodules over a fixed comonad T, towards a fixed target category $\mathcal{E}$.

Remark 37. The family of morphisms constituting a comodule morphism is actually natural with respect to the functoriality defined in Definition 32.

Given a morphism of comonads, we can "transport" comodules over the source comonad to comodules over the target comonad:

Definition 38. *Let $\tau : T \to S$ be a morphism of comonads relative to a functor $F : \mathcal{C} \to \mathcal{D}$, and let furthermore M be a comodule over T towards a category $\mathcal{E}$. We define the **pushforward comodule** $\tau_* M$ to be the comodule over S given by $\tau_* M(A) := MA$ and, for $f : \mathcal{D}(SA, FB)$,*

$$\mathsf{mcobind}^{\tau_* M}(f) := \mathsf{mcobind}^M(f \circ \tau_A) : \mathcal{E}(MA, MB) \ .$$

Pushforward is functorial: if M and N are comodules over T with codomain category $\mathcal{E}$, and $\alpha : M \to N$ is a morphism of comodules, then we define $\tau_* \alpha : \tau_* M \to \tau_* N$ as the family of morphisms $(\tau_* \alpha)_A := \alpha_A$. It is easy to check that this is a morphism of comodules (over S) between $\tau_* M$ and $\tau_* N$. Pushforward thus yields a functor $\tau_* : \mathsf{RComod}(T, \mathcal{E}) \to \mathsf{RComod}(S, \mathcal{E})$.

As presented in Definition 31, every relative comonad induces a comodule over itself. This extends to morphisms of relative comonads:

Definition 39. *Let $\tau : T \to S$ be a morphism of comonads relative to $F : \mathcal{C} \to \mathcal{D}$. Then τ gives rise to a morphism of comodules over S from the pushforward of the tautological comodule of T along τ to the tautological comodule over S,*

$$\langle \tau \rangle : \tau_* T \to S \ , \quad \langle \tau \rangle_A := \tau_A \ .$$

6 Terminality for streams and infinite triangular matrices

In this section, we define a notion of "model" for the signatures of streams and triangular matrices, respectively. We then show that the codata types Stream and Tri constitute the terminal object in the respective category of models.

The terminal semantics result is hardly surprising; however, it is still interesting as it characterizes not only the codata types themselves, but also the respective bisimilarity relations and comonadic operations on them, via a universal property.

6.1 Models for Stream

We first consider the homogeneous codata type of streams.

Definition 40. *A **model for** Stream is given by a pair (S, t) consisting of*

- *a comonad S relative to $\mathsf{eq} : \mathsf{Type} \to \mathsf{Setoid}$ and*
- *a morphism t of tautological comodules over S, $t : S \to S$.*

A model morphism $(S, t) \to (S', t')$ is given by a comonad morphism $\tau : S \to S'$ such that $\langle\tau\rangle \circ \tau_* t = t' \circ \langle\tau\rangle$.

This defines a category, with the obvious composition and identity.

Remark 41 (Coalgebras for streams). We define

$$F : [\mathsf{Type}, \mathsf{Setoid}] \to [\mathsf{Type}, \mathsf{Setoid}], \quad F(G)(A) := \mathsf{eq}(A) \times G(A) \enspace .$$

Then we have a forgetful functor U from models for streams to coalgebras of F which, intuitively, forgets the comonadic structure. Indeed, given a model (T, tail), the comonad T is, in particular, a functor $T : \mathsf{Type} \to \mathsf{Setoid}$ which constitutes the carrier of a coalgebra of F. The coalgebra structure, as a map into a product, is assembled from the counit of T (to give the first component) and the comodule morphism tail, which is a natural transformation $T \to T$ (to give the second component). It is not clear to us whether this forgetful functor has an adjoint.

The reason why we consider the richer notion of model (as compared to coalgebras) is that we want the objects of our category—and thus in particular the terminal object—to be equipped with a well-behaved "cosubstitution" operation. It is this cosubstitution operation which is modeled by the comonad and comodule structure, and which is forgotten by the forgetful functor U defined above.

Theorem 42. *The pair $(\mathsf{Stream}, \mathsf{tail})$, where Stream is considered as a relative comonad and tail as a morphism of comodules, is the terminal object in the category of models of Definition 40.*

More precisely, the aforementioned theorem says that the rules given in Figure 1 allow to prove that the category of models defined in Definition 40 has a terminal object.

Example 43. We equip the relative comonad Tri with the structure of a model for Stream by defining a morphism of tautological comodules over Tri, given by $t^{\mathsf{diag}} := \mathsf{cut} \circ \mathsf{rest} : \mathsf{Tri} \to \mathsf{Tri}$. The resulting terminal model morphism $(\mathsf{Tri}, t^{\mathsf{diag}}) \to (\mathsf{Stream}, \mathsf{tail})$ has as underlying morphism of relative comonads the one defined in Example 26.

Remark 44. Fix a type B. A result analogous to Theorem 42 holds for trees Tree_B of Example 19. We refrain from giving a precise statement of this result.

6.2 Models for Tri

In analogy to the definition of models for the signature of streams, one would define a model for the signature of Tri as a pair (T, r) of a comonad T relative to $\mathsf{eq} : \mathsf{Type} \to \mathsf{Setoid}$ and a morphism of comodules $r : T \to T(E \times \_)$. It turns out that in this way, one is not capable of obtaining the right auxiliary function cut for what is supposed to be the *terminal* such model (where cut is used to define the comodule $\mathsf{Tri}(E \times \_)$), namely the pair $(\mathsf{Tri}, \mathsf{rest})$. As a remedy, we define a model to come equipped with a specified operation analogous to cut, and some laws governing the behavior of that operation:

Definition 45. *Let $\mathcal{C}$ and $\mathcal{D}$ be categories with binary products and $F : \mathcal{C} \to \mathcal{D}$ a product-preserving functor. Let $E : \mathcal{C}_0$ be a fixed object of $\mathcal{C}$. We define a* ***comonad relative to F with cut relative to E*** *to be a comonad T relative to F together with a* cut *operation*

$$\mathsf{cut} : \forall\ A : \mathcal{C}_0, T(E \times A) \to TA \qquad \textit{such that}$$

- $\forall\ A : C_0, \mathsf{counit}_A \circ \mathsf{cut}_A = \mathsf{counit}_A \circ \mathsf{lift}^T(\mathsf{pr}_2(E, A))$*;*
- $\forall\ A\ B : C_0, \forall\ f : \mathcal{D}(TA, FB), \mathsf{cobind}(f) \circ \mathsf{cut}_A = \mathsf{cut}_B \circ \mathsf{cobind}(\mathsf{extend}\ f)$*,*

where, for $f : \mathcal{D}(TA, FB)$, we define $\mathsf{extend}(f) : \mathcal{D}\big(T(E \times A), F(E \times B)\big)$ as

$$\mathsf{extend}(f) := {\phi^F_{E,B}}^{-1} \circ (\mathsf{counit}_E \times f) \circ \langle T(\mathsf{pr}_1), \mathsf{cut}\rangle\ \ .$$

Morphisms of comonads with cut are morphisms of comonads that are compatible with the respective cut operations:

Definition 46. *Let (T, cut^T) and (S, cut^S) be two comonads relative to a functor F with cut relative to E as in Definition 45. A* ***morphism of comonads with cut*** *is a comonad morphism τ between the underlying comonads as in Definition 22 that commutes suitably with the respective* cut *operations, i.e. for any $A : \mathcal{C}_0$, $\mathsf{cut}^S_A \circ \tau_{E \times A} = \tau_A \circ \mathsf{cut}^T_A$.*

Comonads with cut relative to a fixed functor $F : \mathcal{C} \to \mathcal{D}$ and $E : \mathcal{C}_0$ form a category $\mathsf{RComonwCut}(F, E)$. There is a forgetful functor from $\mathsf{RComonwCut}(F, E)$ to $\mathsf{RComon}(F)$. Conversely, any comonad T relative to a suitable functor can be equipped with a cut operation, using functoriality of T.

Remark 47 (Canonical cut *operation).* Any comonad T relative to a product-preserving functor $F : \mathcal{C} \to \mathcal{D}$ can be equipped with a cut operation relative to $E : \mathcal{C}_0$ satisfying the properties of Definition 45 by setting

$$\mathsf{ccut}_A := \mathsf{cut}_A := \mathsf{lift}^T\big(\mathsf{pr}_2(E, A)\big) \enspace .$$

(The extra "c" of ccut stands for "canonical".) It follows from the axioms of comonad morphism that a comonad morphism $\tau : T \to S$ satisfies the equation of Definition 46 for the thus defined operations ccut^T and ccut^S, hence constitutes a morphism of comonads with cut from (T, ccut^T) to (S, ccut^S). We thus obtain a functor

$$\mathsf{ccut}_{F,E} : \mathsf{RComon}(F) \to \mathsf{RComonwCut}(F, E)$$

from relative comonads over F to relative comonads over F with cut relative to a fixed object $E : \mathcal{C}_0$ given on objects by $T \mapsto (T, \mathsf{ccut}^T)$.

The functor $\mathsf{ccut}_{F,E}$, followed by the forgetful functor, yields the identity. We can thus view relative comonads with cut as a generalization of relative comonads.

Our prime example of relative comonad comes with a cut operation that is *not* the canonical one:

Example 48. The relative comonad Tri from Example 20, together with the cut operation defined in Example 10, is a comonad with cut as in Definition 45.

Given a comodule M over a relative comonad T with cut, we define a comodule over T obtained by precomposition of M with "product with a fixed object E":

Definition 49. *Suppose $F : \mathcal{C} \to \mathcal{D}$ is a product-preserving functor, and T is a comonad relative to F with a* cut *operation relative to $E : \mathcal{C}_0$ as in Definition 45. Given a comodule M over T,* ***precomposition with "product** with E"* *gives a comodule $M(E \times \_) : A \mapsto M(E \times A)$ over T. The comodule operation is deduced from that of M by*

$$\begin{aligned} \mathsf{mcobind}^{M(E\times\_)}_{A,B} : \mathcal{D}(TA, FB) &\to \mathcal{E}\big(M(E \times A), M(E \times B)\big) \enspace , \\ f &\mapsto \mathsf{mcobind}^{M}_{E\times A, E\times B}(\mathsf{extend}(f)) \enspace , \end{aligned}$$

where the extend *operation is the one defined in Definition 45.*

Furthermore, given two comodules M and N over $\mathcal{T}$ with target category $\mathcal{E}$, and a comodule morphism $\alpha : M \to N$, the assignment $\alpha(E \times \_)_A := \alpha_{E\times A}$ defines a comodule morphism $\alpha(E \times \_) : M(E \times \_) \to N(E \times \_)$.
We thus obtain an endofunctor on the category of comodules over T towards $\mathcal{E}$, $M \mapsto M(E \times \_) : \mathsf{RComod}(T, \mathcal{E}) \to \mathsf{RComod}(T, \mathcal{E})$.

Remark 50 (Pushforward commutes with product in context). Note that the constructions of Definition 49 and Definition 38 commute: we have an isomorphism of comodules $\tau_*(M(E \times \_)) \cong (\tau_* M)(E \times \_)$ given pointwise by identity morphisms.

It directly follows from the definition that the cut operation of any comonad T with cut constitutes a comodule morphism $\mathsf{cut} : T(E \times \_) \to T$. We can thus restate the definition of a morphism of comonads with cut as in Definition 46 by asking the following diagram of comodule morphisms (in the category $\mathsf{RComod}(S, \mathcal{D})$) to commute (where in the upper left corner we silently add an isomorphism as in Remark 50):

$$\begin{tikzcd}
\tau_* T(E \times \_) \arrow[r, "\tau_*(\mathsf{cut}^T)"] \arrow[d, "\langle\tau\rangle(E\times\_)"'] & \tau_* T \arrow[d, "\langle\tau\rangle"] \\
S(E \times \_) \arrow[r, "\mathsf{cut}^S"'] & S \ .
\end{tikzcd}$$

The construction of Definition 49 yields a categorical characterization of the rest destructor—more precisely, of its behavior with respect to cosubstitution as in Equation (3.2)—via the notion of comodule morphism:

Example 51. This example is a reformulation of Example 36. Consider the comonad Tri, equipped with the cut operation of Example 48. The destructor rest of Example 10 is a morphism of comodules over the comonad Tri from the tautological comodule Tri to $\mathsf{Tri}(E \times \_)$.

Definition 52. *Let $E : \mathsf{Type}$ be a type. Let $\mathcal{T} = \mathcal{T}_E$ be the category of* ***models for infinite triangular matrices*** *where an object consists of*

- *a comonad T over the functor $\mathsf{eq} : \mathsf{Type} \to \mathsf{Setoid}$ with cut relative to E and*
- *a morphism rest of comodules over T of type $T \to T(E \times \_)$*

such that for any set A, $\mathsf{rest}_A \circ \mathsf{cut}_A = \mathsf{cut}_{E\times A} \circ \mathsf{rest}_{E\times A}$.

The last equation can be stated as an equality of comodule morphisms as

$$\mathsf{rest} \circ \mathsf{cut} = \mathsf{cut}(E \times \_) \circ \mathsf{rest}(E \times \_) \quad \big(= (\mathsf{cut} \circ \mathsf{rest})(E \times \_)\big) \ .$$

A morphism between two such objects (T, rest^T) and (S, rest^S) is given by a morphism of relative comonads with cut $\tau : T \to S$ such that the following diagram of comodule morphisms in the category $\mathsf{RComod}(S, \mathcal{E})$ commutes,

$$\begin{tikzcd}
\tau_* T \arrow[r, "\tau_*(\mathsf{rest}^T)"] \arrow[d, "\langle\tau\rangle"'] & \tau_* T(E \times \_) \arrow[d, "\langle\tau\rangle(E\times\_)"] \\
S \arrow[r, "\mathsf{rest}^S"'] & S(E \times \_) \ .
\end{tikzcd}$$

Here we silently insert an isomorphism as in Remark 50 in the upper right corner.

Remark 53 (Coalgebras for Tri). Let $E : \mathsf{Type}$ be a type. We define

$$F : [\mathsf{Type}, \mathsf{Setoid}] \to [\mathsf{Type}, \mathsf{Setoid}], \quad F(G)(A) := \mathsf{eq}(A) \times G(E \times A) \ .$$

Then we have a forgetful functor from models for infinite triangular matrices to coalgebras of F which, intuitively, forgets the comonadic cobind operation and the cut operation. Indeed, given a model $(T, \mathsf{cut}, \mathsf{rest})$, the comonad T is, in particular, a functor $T : \mathsf{Type} \to \mathsf{Setoid}$ which constitutes the carrier of a coalgebra of F. The coalgebra structure on T, being a map into a product, is assembled from the counit of T (to give the first component) and the comodule morphism $\mathsf{rest} : T \to T(E \times \_$ (to give the second component). Note that both the counit and rest are natural transformations. It is not clear to us whether this forgetful functor has an adjoint.

Theorem 54. *The pair* $(\mathsf{Tri}, \mathsf{rest})$ *consisting of the relative comonad with cut* Tri *of Example 48 together with the morphism of comodules* rest *of Example 36, constitutes the terminal model of triangular matrices.*

Proof (sketch). For a given model (T, rest^T), the (terminal) morphism $\bigcirc = \bigcirc_T : T \to \mathsf{Tri}$ is defined via the corecursive equations

$$\mathsf{top}\big(\bigcirc\ t\big) := \mathsf{counit}^T\ t \quad \text{and} \tag{6.1}$$

$$\mathsf{rest}\big(\bigcirc\ t\big) := \bigcirc(\mathsf{rest}^T\ t)\ . \tag{6.2}$$

By coinduction we show that the map $\bigcirc$ is compatible with cobind and cut operations of the source and target models. We omit these calculations, which can be consulted in the `Coq` source files.

Note that there is actually no choice in this definition: Equation (6.1) is forced upon us since we want $\bigcirc$ to constitute a morphism of comonads—the equation directly corresponds to one of the axioms. Equation (6.2) is forced upon us by the diagram a morphism of models has to make commute.

The same argument is used to show, again by coinduction, that any two morphisms of models $\tau, \rho : (T, \mathsf{rest}^T) \to (\mathsf{Tri}, \mathsf{rest})$ are equal, thus concluding the proof.

This universal property characterizes not only the codata type Tri, but also the **bisimilarity** relation on it as well as the **cosubstitution** operation.

7 Formalization in `Coq`

All our definitions and theorems are mechanized in the proof assistant `Coq` [9]. In the formalization we add the rules of Figure 1 and Figure 3 as axioms internal to the `Coq` system, via the `Axiom` vernacular. In order to ensure consistency, we

1. encapsulate the axioms in `Coq` *module* types (separately for Stream and for Tri) and
2. instantiate each module type by defining streams and triangular matrices as coinductive types using the `CoInductive` vernacular, as shown in Figure 4 for the example of Tri.

```
CoInductive Tri A : Type :=
   constr : A -> Tri (E x A) -> Tri A.
CoInductive bisim : forall {A}, Tri A -> Tri A -> Prop :=
   bisim_constr : forall {A} {t t' : Tri A},
        top t = top t' ->
        bisim (rest t) (rest t') ->
        bisim t t'.
```

Here, the code `E x A` denotes the cartesian product of the types `E` and `A`.

Fig. 4. Definition of Tri and bisimilarity using `Coq`'s `CoInductive` vernacular

This shows that the axioms we add to `Coq` are weaker than the general mechanism of defining coinductive types in `Coq` via the `CoInductive` vernacular.

For the instantiation of the module type for Tri, the formalization of infinite triangular matrices is taken from the work by Matthes and Picard [18], and only slightly adapted to compile with the version of `Coq` we use.

Apart from the implementation of our codata types via axioms, the mechanization does *not* rely on any additional axioms. The `Coq` source files and HTML documentation are available from the project web site [5]. The correspondence between formalized statements and the statements in this article are given in Figure 5.

Informal	Reference	Formal
Category		`Category`
Functor		`Functor`
Relative comonad	Def. 12	`RelativeComonad`
Triangular matrices as comonad	Ex. 20	`Tri`
Comodule over comonad	Def. 30	`Comodule`
Tautological comodule (of T)	Def. 31	`tcomod`, `<T>`
tail is comodule morphism	Ex. 35	`Tail`
rest is comodule morphism	Ex. 36	`Rest`
Pushforward comodule	Def. 38	`pushforward`
Induced comodule morphism	Def. 39	`induced_morphism`
Models for streams	Def. 40	`Stream`
Stream is terminal	Thm. 42	`StreamTerminal.Terminality`
Relative comonad with cut	Def. 45	`RelativeComonadWithCut`
Precomposition with product	Def. 49	`precomposition_with_product`
Models for triangular matrices	Def. 52	`TriMat`
Tri is terminal	Thm. 54	`TriMatTerminal.Terminality`

Fig. 5. Correspondence of informal and formal definitions

8 Future work

In a forthcoming work, we develop a notion of signature for specifying codata types with a cosubstitution operation, and give a categorical semantics for such comonadic codata types.

In a more extensional type theory, such as *Homotopy Type Theory* [22], one can reconcile bisimilarity and propositional equality, thus eliminating the need to work with setoids. This will be investigated in future work.

Acknowledgments We thank André Hirschowitz, Ralph Matthes and Paige North for many helpful discussions.

References

[1] Andreas Abel, Ralph Matthes, and Tarmo Uustalu. "Iteration and coiteration schemes for higher-order and nested datatypes". In: *Theor. Comput. Sci.* 333.1-2 (2005), pp. 3–66.

[2] Andreas Abel, Brigitte Pientka, David Thibodeau, and Anton Setzer. "Copatterns: programming infinite structures by observations". In: *Principles of Programming Languages*. Ed. by Roberto Giacobazzi and Radhia Cousot. ACM, 2013, pp. 27–38.

[3] Peter Aczel. *Galois: A Theory Development Project*. Technical Report for the 1993 Turin meeting on the Representation of Mathematics in Logical Frameworks. 1993. URL: http://www.cs.man.ac.uk/~petera/papers.html.

[4] Benedikt Ahrens. *Modules over relative monads for syntax and semantics*. Accepted for pub. in Math. Struct. in Comp. Science. arXiv:1107.5252.

[5] Benedikt Ahrens and Régis Spadotti. *Terminal semantics for codata types in intensional Martin-Löf type theory*. http://benediktahrens.github.io/coinductives/. arXiv:1401.1053.

[6] Thorsten Altenkirch, James Chapman, and Tarmo Uustalu. "Monads Need Not Be Endofunctors". In: *Foundations of Software Science and Computational Structures*. Ed. by C.-H. Luke Ong. Vol. 6014. Lecture Notes in Computer Science. Springer, 2010, pp. 297–311.

[7] Thorsten Altenkirch and Bernhard Reus. "Monadic presentations of lambda terms using generalized inductive types". In: *Computer Science Logic, 13th International Workshop*. 1999, pp. 453–468.

[8] Steven Awodey, Nicola Gambino, and Kristina Sojakova. "Inductive Types in Homotopy Type Theory". In: *LICS*. IEEE, 2012, pp. 95–104.

[9] Coq development team. *The Coq Proof Assistant, v8.4pl3*. 2013.

[10] Thierry Coquand. "Infinite Objects in Type Theory". In: *TYPES*. Ed. by Henk Barendregt and Tobias Nipkow. Vol. 806. Lecture Notes in Computer Science. Springer, 1993, pp. 62–78.

[11] Peter Dybjer. "Representing Inductively Defined Sets by Wellorderings in Martin-Löf's Type Theory". In: *Theor. Comput. Sci.* 176.1-2 (1997), pp. 329–335.

[12] Marcelo Fiore, Gordon Plotkin, and Daniele Turi. "Abstract Syntax and Variable Binding". In: *Proceedings of the 14th Annual IEEE Symposium on Logic in Computer Science*. Washington, DC, USA: IEEE Computer Society, 1999, pp. 193–202.

[13] André Hirschowitz and Marco Maggesi. "Modules over monads and initial semantics". In: *Inf. Comput.* 208.5 (2010), pp. 545–564.

[14] Gérard Huet and Amokrane Saïbi. "Constructive Category Theory". In: *In Proceedings of the Joint CLICS-TYPES Workshop on Categories and Type Theory, Goteborg*. MIT Press, 1998.

[15] Bart Jacobs and Jan Rutten. "A tutorial on (co) algebras and (co) induction". In: *Bulletin-European Association for Theoretical Computer Science* 62 (1997), pp. 222–259.

[16] Federico De Marchi. "On the Semantics of Coinductive Types in Martin-Löf Type Theory". In: *CALCO*. Ed. by José Luiz Fiadeiro, Neil Harman, Markus Roggenbach, and Jan J. M. M. Rutten. Vol. 3629. Lecture Notes in Computer Science. Springer, 2005, pp. 114–126.

[17] Per Martin-Löf. *Intuitionistic Type Theory*. Bibliopolis, 1984.

[18] Ralph Matthes and Celia Picard. "Verification of redecoration for infinite triangular matrices using coinduction". In: *Workshop on Types for Proofs and Programs*. Ed. by Nils Anders Danielsson and Bengt Nordström. Vol. 19. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2011, pp. 55–69.

[19] Ieke Moerdijk and Erik Palmgren. "Wellfounded trees in categories". In: *Ann. Pure Appl. Logic* 104.1-3 (2000), pp. 189–218.

[20] John Power. "Abstract Syntax: Substitution and Binders". In: *Electron. Notes Theor. Comput. Sci.* 173 (Apr. 2007), pp. 3–16.

[21] Miki Tanaka and John Power. "A unified category-theoretic formulation of typed binding signatures". In: *Proceedings of the 3rd ACM SIGPLAN workshop on Mechanized reasoning about languages with variable binding*. MERLIN '05. Tallinn, Estonia: ACM, 2005, pp. 13–24.

[22] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. Institute for Advanced Study: http://homotopytypetheory.org/book, 2013.

[23] Tarmo Uustalu and Varmo Vene. "The Dual of Substitution is Redecoration". In: *Scottish Functional Programming Workshop*. Ed. by Kevin Hammond and Sharon Curtis. Vol. 3. Trends in Functional Programming. Intellect, 2001, pp. 99–110.