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, France

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`.

1998 ACM Subject Classification F.3.2 Semantics of Programming Languages

Keywords and phrases relative comonad, Martin-Löf type theory, coinductive type, computer theorem proving

Digital Object Identifier 10.4230/LIPIcs.TYPES.2014.1

1 Introduction

In this work, we study the notions of *relative comonad* and *comodule over a relative comonad.* We then use these notions for a case study in categorical semantics of coinductive data types in intensional Martin-Löf type theory (IMLTT): we characterize two coinductive type families and their respective cosubstitution operations in IMLTT via a universal property. The first codata type we consider is the *homogeneous* type family of streams, parametrized by a base type. The second one is 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: in Section 1.1 we briefly introduce (intensional) Martin-Löf type theory. In Section 1.2 we discuss inductive types and their semantics, before passing to *co*inductive types in Section 1.3. We describe the difference between homogeneous and heterogeneous (co)data types in Section 1.4 and explain substitution for leaf-labeled trees and cosubstitution for node-labeled trees in Section 1.5. Sections 1.6 to 1.8 of the introduction concern more "administrative" aspects.

1.1 Intensional Martin-Löf type theory

Martin-Löf type theory (MLTT) [17] is a dependent type theory developed in the 1970's, which provides a foundation of mathematics. It is based on the Curry-Howard isomorphism, that is, it treats logic as a fragment of the general type theory. There are two notions of "sameness" in Martin-Löf type theory, an external (judgmental) one, and an internal (propositional) one. The latter is given by the *Martin-Löf identity type*, which is an inductively defined binary relation on any given type. Its only constructor relates any term to itself, that is, the relation defined by the Martin-Löf identity type is the least reflexive relation on

[*] The work of Benedikt Ahrens was partially supported by the CIMI (Centre International de Mathématiques et d'Informatique) Excellence program ANR-11-LABX-0040-CIMI within the program ANR-11-IDEX-0002-02 during a postdoctoral fellowship.

a given type. Judgmentally equal terms are always propositionally equal, but the converse is not always true. Indeed, 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. *Intensional* Martin-Löf type theory lacks the reflection principle for the sake of decidability of type checking. This variant forms the basis of the computer proof assistants `Coq`, `Matita` and `Agda`.

1.2 Wellfounded trees and their semantics

A tree is a special kind of graph (with labeled edges and unlabeled vertices), consisting of a *root* (edge), to which a number of *subtrees* are attached. The subtrees are themselves trees. The root of a tree and the roots of any subtrees are more generally called *nodes*. We gather trees of a fixed shape into a set or type, depending on the foundations we work in. The possible shapes of trees are given by two pieces of data:

1. the set/type of potential nodes and
2. the number of subtrees attached to each node.

Such a pair is called a *signature*, and we are interested in the set/type of trees which are formed according to a given signature.

A tree is called *wellfounded* if all of its subtrees have finite length. In this section we consider the set/type of wellfounded trees of a given signature. Such a set/type is called *inductively generated* by the signature. Intuitively, this set/type is the least one stable under forming trees starting from leaves and attaching already constructed trees as subtrees to nodes. One might also consider *potentially infinite*, i.e., non-wellfounded trees; these are called *co*inductive and are the main object of this work. We discuss them in Section 1.3.

Many objects in mathematics and logic can be represented as (wellfounded) *trees*. For instance, a natural number can be considered as such a tree: let $\{S, Z\}$ be the set of possible nodes, such that node S always has one subtree and node Z does not have any subtrees, i.e. Z is a *leaf*. The natural number 2 is represented by the tree $S \longrightarrow S \longrightarrow Z$. This example demonstrates the fundamental aspect of those tree-like objects: they can be used to model basic mathematical and logical objects.

By "semantics of inductive types" we mean a mathematical description of the set/type of trees of a given signature. More precisely, the goal is to give a category-theoretic explanation to the above description as "the least set/type stable under forming trees". However, the question of how to give such a characterization depends on the formal system we work in.

In a *set-theoretic* setting, inductive sets are characterized as initial algebras for some endofunctor on the category of sets [16].

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 (see, e.g., [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., [12, 9]). 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.

The reflection rule equips *extensional* type theory with extensional features similar to those of set theory. As a consequence, the characterization given by Dybjer [12] of an inductive type in extensional MLTT works as in set theory: an inductive type constitutes the initial algebra for some endofunctor on the category of types.

In *intensional* Martin-Löf type theory the characterization of inductive types as initial algebras of some endofunctor fails for a lack of function extensionality [12], but it can be recovered [9] in an *extension* of intensional MLTT called *Homotopy Type Theory* (HoTT) [23]. In this extension, function extensionality is provable from the *Univalence Axiom*. For a suitable definition of *uniqueness* – *contractibility* in HoTT jargon – one can prove a logical equivalence between type-theoretic rules specifying an inductive type on the one hand and the existence of an initial algebra within the theory on the other hand. The mentioned work [9] 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.

1.3 Non-wellfounded trees: simply dualizing?

Coinductive sets/types describe sets/types of potentially *infinite* trees, that is, trees that may have an infinite chain of subtrees – still formed according to some signature. An example of such a coinductive set/type is given by the *co*natural numbers, which are specified by the same signature as the natural numbers. They consist of the elements of the natural numbers and an additional, infinite, tree $S \longrightarrow S \longrightarrow S \longrightarrow \ldots$.

In a traditional set-theoretic setting, the theory of coinductive sets is completely dual to that of inductive sets: *co*inductive sets are characterized as terminal coalgebras of suitable endofunctors [16].

In intensional Martin-Löf 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 IMLTT modulo identity, one defines a binary coinductive relation called *bisimilarity* on a given coinductive type, with respect to which one compares its inhabitants [11]. Expressed categorically, bisimilarity in IMLTT is given as a *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 HoTT.

With these conventions, we give, in the present work, a characterization of some coinductive data types as *terminal* object in some category defined in IMLTT. 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 IMLTT, 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 the considered codata types are endowed in a canonical way. The cosubstitution operation is explained in Section 1.5.

In the present work, we do not study *existence* of coinductive types. In variants of IMLTT, coinductive types have been derived from inductive types: for IMLTT with Uniqueness of Identity Proofs [7], and for IMLTT augmented by the Univalence Axiom [4].

1.4 Homogeneous and heterogeneous trees

Instead of considering types of trees of a given signature, we consider, more generally, *families* $T(X)$ of types of trees, families parametrized by a type X. We call a tree $t : T(X)$ over X *homogeneous* if all of its subtrees are also trees over X, i.e. if all the subtrees are elements of $T(X)$. A signature accordingly is called homogeneous if it only admits homogeneous trees. On the other hand, a tree t is called *heterogeneous*, if its subtrees t_i ($i : I$ for some indexing type I) are trees over type $F_i(X)$ for maps of types $F_i : \mathsf{Type} \to \mathsf{Type}$. Obviously, when the functors F_i are identity functors, we get back homogeneous trees; heterogeneous trees are thus more general than homogeneous trees.

In the present work, we study examples of both homogeneous and heterogeneous trees: a *stream* is a *homogeneous* tree with one subtree over the same type of nodes, whereas an *infinite triangular matrix* is a *heterogeneous* tree (over a type, say, A) with one subtree over type $E \times A$ for a fixed, but arbitrary type E. This is explained in more detail below.

The examples we consider are trees with just one subtree. A generalization to trees (no matter whether homogeneous or heterogeneous) with a family of subtrees indexed by some fixed type I as above is straightforward – we refrain from striving for this additional generality in order to keep the exposition as simple and clear as possible, and also in view of a lack of practical examples where this generality would be useful.

1.5 Cosubstitution: a comonadic operation on parametrized codata

We first consider the case of *leaf-labeled trees.* Let $T(X)$ be a type family of trees (of a given shape) with *leaves from* X, i.e., $t : T(X)$ is a tree (of a given shape) with leaves in X. Let furthermore $f : X \to T(Y)$ be a map. We obtain a tree $t' : T(Y)$ by replacing any leaf of t by its image under the map f. This replacement operation is called *(simultaneous) substitution* – "simultaneous" because all the leaves are substituted at once. A well-studied example is that of the lambda calculus, the terms of which can be formalized as leaf-labeled trees over a type of *free variables*, that is, over a *context* [8]. More precisely, let $\mathsf{LC}(X)$ be the type of lambda terms in context X. A map $f : X \to \mathsf{LC}(Y)$ is called a *substitution rule*, indicating how to substitute any free variable $x : X$ occurring in a term $t : \mathsf{LC}(X)$. The substitution operation (parametrized by contexts X and Y) takes as input a lambda term $t : \mathsf{LC}(X)$ and a rule $f : X \to \mathsf{LC}(Y)$ and returns the substituted term $t' : \mathsf{LC}(Y)$. The categorical characterization of this substitution operation has been studied extensively; our list of pointers [8, 13, 20, 22, 14, 6, 3] 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 [8, 14, 6, 3].

Node-labeled trees can 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 [24].

If we consider infinite (i.e., coinductive) node-labeled trees in IMLTT, however, the algebraic laws of cosubstitution for such trees have to be considered *modulo bisimilarity rather than identity.* For this we develop the notion of *relative comonad* and *comodule over a relative comonad.* Indeed, we show that our exemplary codata type families of streams and triangular matrices constitute relative comonads, and the maps which return the substream resp. submatrix of a given stream resp. matrix constitute comodule morphisms over those relative comonads.

Our goal is to characterize those codata types and their respective cosubstitution operation as a universal object in some category. Traditionally, codata types are characterized as terminal coalgebras of some endofunctor. However, this characterization does not account for the cosubstitution operation. In order to integrate cosubstitution into the categorical semantics of the coinductive types under consideration, we thus define a more complicated category of "models" of the signature specifying those types – coalgebras with extra structure accounting for a comonadic operation. The terminal such model is given by the codata type together with its cosubstitution operation. Our terminal semantics thus 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.

1.6 Computer-checked proofs

All our results have been implemented in the proof assistant `Coq` [10]. The `Coq` source files and HTML documentation are available online [5]. Here, we hence omit the proofs and focus on definitions and statements.

1.7 Type theory vs. set theory

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.

1.8 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, in particular, the respective cosubstitution operations. In Section 4 we present *relative comonads* and define the category of comonads relative to a fixed functor. We give some examples of relative comonads using the codata types presented in Section 3, and relate relative comonads to traditional comonads. In Section 5 we define *comodules over relative comonads* and give some constructions of comodules. Again, examples of comodules are taken from Section 3. In Section 6 we define categories of models for the (signatures of the) codata types presented in Section 3, based on the category-theoretic notions developed in the preceding sections. We then prove that the codata types constitute the terminal models in the respective categories. We present an example of a map defined as a terminal morphism in the category of models for streams, exploiting terminality of Stream. In Section 7 we give an overview of the formalization of this work in the proof assistant `Coq`.

2 Preliminaries

We present a few particular category-theoretic objects used later on, and fix some notation.

▶ **Definition 1** (Setoids in IMLTT). 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** (Category in IMLTT, [2, 15]). 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.

We write $f : a \to b$ for $f : \mathcal{C}(a, b)$, 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** (Some categories in IMLTT). We denote by Type the category (in the sense of Definition 2) 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, $f \sim g \ =_{\mathrm{def}}\ \forall x : A, fx = gx$.

We denote by Setoid the category (in the sense of Definition 2) 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 equivalence relation on setoid morphisms is given by pointwise equivalence: two parallel morphisms of setoids $f, g : A \to B$ are equivalent if for any $a : A$ we have $fa \sim_B ga$. The category Setoid thus is cartesian closed.

▶ **Definition 4** (A functor from types to setoids). 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, i.e., binary products and a terminal object.

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

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

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

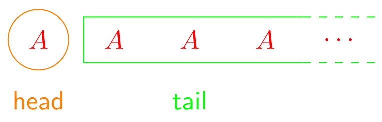


Figure 1 A stream decomposes into its head and tail.

3 Codata types in intensional Martin-Löf type theory

In this section, two coinductive type families in intensional Martin-Löf type theory (IMLTT) [17] are presented. For $a, b : A$, the identity type between a and b is denoted by $a = b$.

We present the type-theoretic rules specifying 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 (in a sense to be specified later) with the *observations* on that codata; it is the appropriate notion of sameness on inhabitants of these types [11]. Here, an observation is the result of *destructing*, i.e. taking apart, an element of the codata type. For instance, an infinite list a.k.a. a stream (defined in detail in Example 8) over a base type A allows to observe the first element – the head – of that list (which is a term of type A) and, corecursively, the tail of that list, which is again an infinite list over A.

A coinductive type together with its associated bisimilarity relation forms a setoid as in Definition 3. We thus denote bisimilar elements t and t' by $t \sim t'$.

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 2. We thus use a kind of "copattern matching" [1] to define maps into streams (and analogously for other codata types): the definition $f := \mathsf{corec}\ h\ t$ is written as the pair of clauses

$$\mathsf{head} \circ f := h \qquad \text{and} \qquad \mathsf{tail} \circ f := t \ .$$

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

▶ **Example 8** (The coinductive type family of streams)**.** Let A be a type. A *stream over* A is, intuitively, an infinite list of elements of A. Given such a stream t, we can extract an element of A, called $\mathsf{head}(t)$ and the rest of the stream, called $\mathsf{tail}(t)$, which is again a stream over A; see Figure 1 for a schematic illustration of the two operations.

Formally, the type – setoid, actually – $\mathsf{Stream}A$ of *streams over* A is coinductively defined via the destructors head and tail given in Figure 2. A map $f : T \to \mathsf{Stream}A$ from an arbitrary type T into the type of streams is specified by giving, for each possible input $t : T$, an element $a : A$ – the head of the image stream – and another $t' : T$ – an element which is mapped to tail $f(t)$. This intuition is made formal by the introduction and computation rules of Figure 2.

The second part of Figure 2 introduces *bisimilarity* as a binary relation on streams (over a fixed type A) that is compatible with the observations in the sense that bisimilar streams have equal heads and bisimilar tails (cf. the elimination rules). We call a *bisimulation on streams (over a given type)* any binary relation that is compatible with the observations on streams. If two streams are related by *any* bisimulation, then they are also bisimilar; this is what the introduction rule declares. In that sense, bisimilarity is the largest bisimulation.

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

$$\frac{\begin{array}{r@{}l} & \vdash R : \mathsf{Stream}A \to \mathsf{Stream}A \to \mathsf{Type} \\ x, y : \mathsf{Stream}A & {} \vdash R\ x\ y \to \mathsf{head}\ x = \mathsf{head}\ y \\ x, y : \mathsf{Stream}A & {} \vdash R\ x\ y \to R\ (\mathsf{tail}\ x)(\mathsf{tail}\ y) \end{array}}{x, y : \mathsf{Stream}A \vdash R\ x\ y \to \mathsf{bisim}\ x\ y}$$

Figure 2 Rules for streams and bisimilarity on them.

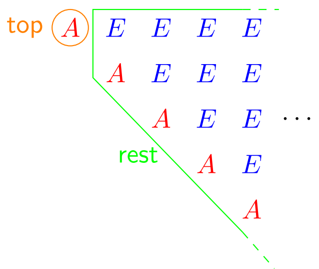


Figure 3 An infinite triangular matrix over type A and its destructors top and rest.

We define a *co*substition operation on streams via the following clauses:

$$\begin{aligned} \mathsf{cosubst}_{A,B} &: (\mathsf{Stream} A \to B) \to \mathsf{Stream} A \to \mathsf{Stream} B \\ \mathsf{head} \circ \mathsf{cosubst}\ f &:= f \quad \text{and} \\ \mathsf{tail} \circ \mathsf{cosubst}\ f &:= \mathsf{cosubst}\ f \circ \mathsf{tail} \ . \end{aligned}$$

According to our convention above, by this, we actually mean

$$\mathsf{cosubst}_{A,B}(f) := \mathsf{corec}\ f\ \mathsf{tail} \ .$$

We call this operation "cosubstitution" since its type is dual to the simultaneous substitution operation of the lambda calculus [8]. 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 type analogous to that defining tail of Example 8. We thus obtain Stream by considering, for B, the singleton type.

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

▶ **Example 10.** Infinite triangular matrices are studied in detail by Matthes and Picard [18]. 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 3.

The codata type is specified via two destructors top and rest, whose types are given in Figure 4.

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 3, 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, shown in Figure 5. 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 the reason why the family Tri is called *heterogeneous*.

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

$$\frac{\begin{array}{c} A : \mathsf{Type} \vdash R : \mathsf{Tri}A \to \mathsf{Tri}A \to \mathsf{Type} \\ A : \mathsf{Type}, x, y : \mathsf{Tri}A \vdash R\ x\ y \to \mathsf{top}\ x = \mathsf{top}\ y \\ A : \mathsf{Type}, x, y : \mathsf{Tri}A \vdash R\ x\ y \to R\ (\mathsf{tail}\ x)(\mathsf{tail}\ y) \end{array}}{A : \mathsf{Type}, x, y : \mathsf{Tri}A \vdash R\ x\ y \to \mathsf{bisim}\ x\ y}$$

Figure 4 Rules for triangular matrices and bisimilarity on them.

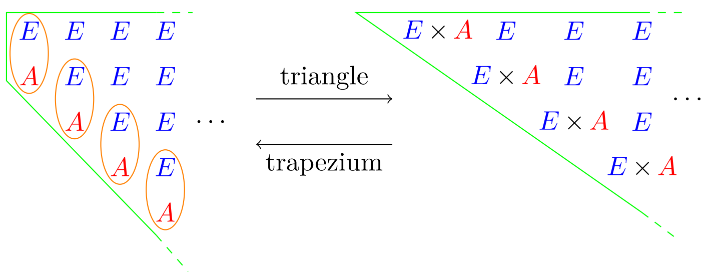


Figure 5 The rest of a matrix over A is a matrix over $E \times A$ (illustration taken from [18]).

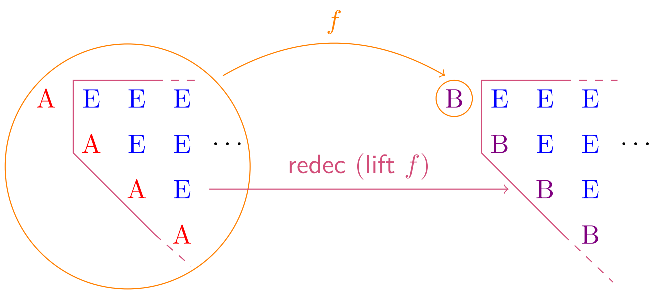


Figure 6 Definition of redecoration (illustration taken from [18]).

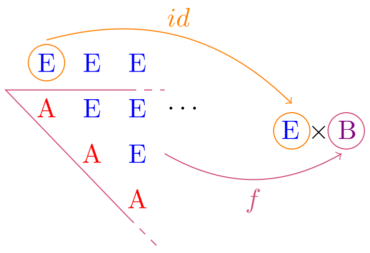


Figure 7 Definition of lifting (illustration taken from [18]).

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

$$\mathsf{top} \circ \mathsf{redec}\ f := f \quad \text{and} \quad \mathsf{rest} \circ \mathsf{redec}\ f := \mathsf{redec}\ (\mathsf{lift}\ f) \circ \mathsf{rest}\ . \tag{1}$$

An illustration of the redecoration operation is given in Figure 6 (taken from [18]).

Here, the need for the family of auxiliary functions

$$\mathsf{lift}_{A,B} : (\mathsf{Tri}A \to B) \to \mathsf{Tri}(E \times A) \to E \times B$$

arises from the *heterogeneity* of the destructor rest. These functions are suitably defined to account for the change of the type of the argument of redec when redecorating rest $t : \mathsf{Tri}(E\times A)$ rather than $t : \mathsf{Tri}A$, namely

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

An illustration of this operation is given in Figure 7 (taken from [18]).

The auxiliary function $\mathsf{cut}_A : \mathsf{Tri}(E \times A) \to \mathsf{Tri}A$ cuts the upper row of elements in E of a trapezium (equivalently, of a matrix over $E \times A$). It is defined corecursively via

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

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

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

▶ **Remark 11.** *We do not specify any computation rules for bisimilarity, i.e., we think of bisimilarity as a proof-irrelevant notion. The lack of such a computation rule amounts to asserting that bisimilarity is a* weakly *terminal bisimulation on the respective codata.*

4 Relative comonads and their morphisms

In this section we define *comonads relative to a functor* and morphisms between those, and present some examples.

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. An example of relative monad given in that work is the lambda calculus over finite contexts.

The dual notion is that of a relative *co*monad, more precisely, a comonad relative to some functor $F : \mathcal{C} \to \mathcal{D}$. Indeed, since the functor underlying a relative comonad is not necessarily endo, one requires an auxiliary functor, which "mediates" between source and target category. An application of this auxiliary functor is inserted where necessary to make the comonad-like operations and axioms welltyped.

▶ **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 and
- operations

$$\begin{aligned}\mathsf{counit} &: \forall A : \mathcal{C}_0, \mathcal{D}(TA, FA)\\ \mathsf{cobind} &: \forall A, B : \mathcal{C}_0, \mathcal{D}(TA, FB) \to \mathcal{D}(TA, TB)\end{aligned}$$

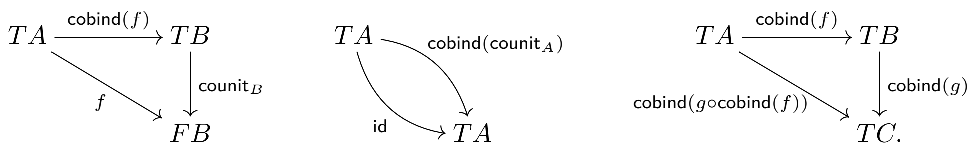
such that the following equations hold:

$$\forall A, B : \mathcal{C}_0, \forall f : \mathcal{D}(TA, FB), \mathsf{counit}_B \circ \mathsf{cobind}(f) = f \tag{2}$$

$$\forall A : \mathcal{C}_0, \mathsf{cobind}(\mathsf{counit}_A) = \mathsf{id}_{TA} \tag{3}$$

$$\forall A, B, C : \mathcal{C}_0, \forall f : \mathcal{D}(TA, FB), \forall g : \mathcal{D}(TB, FC),\quad \mathsf{cobind}(g) \circ \mathsf{cobind}(f) = \mathsf{cobind}(g \circ \mathsf{cobind}(f))\ , \tag{4}$$

in diagrammatic form:


$$\begin{tikzcd} TA \arrow[r, "\mathsf{cobind}(f)"] \arrow[dr, "f"'] & TB \arrow[d, "\mathsf{counit}_B"] \\ & FB \end{tikzcd} \qquad \begin{tikzcd} TA \arrow[r, bend left, "\mathsf{cobind}(\mathsf{counit}_A)"] \arrow[r, bend right, "\mathsf{id}"'] & TA \end{tikzcd} \qquad \begin{tikzcd} TA \arrow[r, "\mathsf{cobind}(f)"] \arrow[dr, "\mathsf{cobind}(g\circ\mathsf{cobind}(f))"'] & TB \arrow[d, "\mathsf{cobind}(g)"] \\ & TC. \end{tikzcd}$$

Definition 12 does not mention an action of T on morphisms. Indeed, just like with relative monads, this action is definable rather than part of the definition:

▶ **Definition 13.** Let T be a comonad relative to $F : \mathcal{C} \to \mathcal{D}$. For $f : \mathcal{C}(A, B)$ we define

$$T(f) := \mathsf{cobind}(Ff \circ \mathsf{counit}_A) : \mathcal{D}(TA, TB)\ .$$

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:*

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

Comonads relative to the identity functor are exactly (traditional) comonads.

We obtain a comonad relative to $F : \mathcal{C} \to \mathcal{D}$ from a comonad on the category $\mathcal{C}$ by composing M with F:

▶ **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:

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

The other way round does not require any conditions on the functor F:

▶ **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:

$$\begin{aligned}MF(A) &:= M(FA)\\ \mathsf{counit}^{MF}_A &:= \mathsf{counit}^M_{FA} : \mathcal{D}(MFA, FA)\\ \mathsf{cobind}^{MF}_{A,B}(f) &:= \mathsf{cobind}^M_{FA,FB}(f) : \mathcal{D}(MFA, MFB) \quad \text{for} \quad f : \mathcal{D}(MFA, FB)\ .\end{aligned}$$

The next examples concern the codata type families presented in Section 3.

▶ **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* $\mathsf{eq} : \mathsf{Type} \to \mathsf{Setoid}$*.*

▶ **Remark 22** (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 $\mathcal{D}$,*

$$\mathsf{cobind}_{A,B} : \underline{\mathcal{D}}(TA, FB) \to \underline{\mathcal{D}}(TA, TB) \ . \tag{5}$$

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* (5) *encodes a higher-order compatibility of cosubstitution with bisimilarity, namely that cosubstitution with two extensionally bisimilar functions yields extensionally bisimilar functions.*

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 (for streams) resp. rest (for infinite triangular matrices), we develop the notion of *comodule over a relative comonad* and, more importantly, their morphisms, in Section 5.

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

▶ **Definition 23.** 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

$$\tau_A : TA \to SA$$

such that the following diagrams commute for any $A, B : \mathcal{C}_0$ and $f : SA \to FB$:

$$\begin{tikzcd} TA \arrow[r, "\tau_A"] \arrow[dr, "\mathsf{counit}^T_A"'] & SA \arrow[d, "\mathsf{counit}^S_A"] \\ & FA \end{tikzcd} \qquad \begin{tikzcd} TA \arrow[r, "\mathsf{cobind}^T(f \circ \tau_A)"] \arrow[d, "\tau_A"'] & TB \arrow[d, "\tau_B"] \\ SA \arrow[r, "\mathsf{cobind}^S(f)"'] & SB. \end{tikzcd}$$

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

▶ **Remark 24.** *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 25** (Example 15 continued)**.** Let M and M' be two comonads on $\mathcal{C}$, and let $\tau : M \to M'$ be a comonad morphism. The family of morphisms

$$F\tau_A := F(\tau_A) : FMA \to FM'A$$

constitutes a morphism of relative comonads $F\tau : FM \to FM'$.

▶ **Remark 26.** *The definitions given in Examples 15 and 25 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.*

An example of comonad morphism is given by the *diagonal* map which extracts the diagonal of an infinite triangular matrix (over type A), yielding a stream over A:

▶ **Example 27.** 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

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

Instead of proving the comonad morphism axioms for this map, we will, in Example 45, specify the same map via a universal property. There, the compatibility of this map with cosubstitution on source and target will follow for free.

▶ **Remark 28** (Non-examples)**.** *The family of destructors* $\mathsf{tail}_A : \mathsf{Stream}A \to \mathsf{Stream}A$ *does* not *satisfy the axioms for a morphism of relative comonads* $\mathsf{Stream} \to \mathsf{Stream}$.

Similarly, while the map $A \mapsto \mathsf{Tri}(E \times A)$ *inherits the structure of a relative comonad (see Definition 29), the family* $\mathsf{rest}_A : \mathsf{Tri}A \to \mathsf{Tri}(E \times A)$ *does* not *constitute a comonad morphism of type* $\mathsf{Tri} \to \mathsf{Tri}(E \times \_)$.

Let $\mathcal{C}$ be a category with products and $E : \mathcal{C}_0$ be an object of $\mathcal{C}$. The following definition shows how a relative comonad T with domain category $\mathcal{C}$ gives rise to a relative comonad with underlying object map $A \mapsto T(E \times A)$. We shall not make use of this definition in what follows: indeed, in Section 6 we consider the map $A \mapsto T(E \times A)$ as a *comodule* over T, rather than a comonad. The below definition may thus be skipped by the reader.

▶ **Definition 29.** 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

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

and the cobind operation as

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

with lift' defined as

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

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 [3]. Our motivation for developing this notion is the characterization of the destructors tail and rest as *morphisms of comodules*.

▶ **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 the following equations hold:

$$\forall A : \mathcal{C}_0, \mathsf{mcobind}(\mathsf{counit}_A) = \mathsf{id}_{MA} \tag{6}$$

$$\begin{aligned}&\forall A, B, C : \mathcal{C}_0, \forall f : \mathcal{D}(TA, FB), \forall g : \mathcal{D}(TB, FC),\\ &\quad \mathsf{mcobind}(g) \circ \mathsf{mcobind}(f) = \mathsf{mcobind}(g \circ \mathsf{cobind}(f)) \ ,\end{aligned} \tag{7}$$

in diagrammatic form:

$$\begin{tikzcd} MA \arrow[dr, bend left, "\mathsf{mcobind}(\mathsf{counit}_A)"] \arrow[dr, bend right, "\mathsf{id}"'] & \\ & MA \end{tikzcd} \qquad \begin{tikzcd} MA \arrow[r, "\mathsf{mcobind}(f)"] \arrow[dr, "\mathsf{mcobind}(g \circ \mathsf{cobind}(f))"'] & MB \arrow[d, "\mathsf{mcobind}(g)"] \\ & MC. \end{tikzcd}$$

Similarly to relative comonads, comodules over these are functorial:

▶ **Definition 31** (Functoriality for comodules)**.** 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

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

Every relative comonad constitutes a comodule over itself, the *tautological* comodule:

▶ **Definition 32** (Tautological comodule)**.** 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 $\mathsf{mcobind}^T(f) := \mathsf{cobind}^T(f)$.

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 mcobind as (cf. Example 10)

$$\mathsf{mcobind}_{A,B}(f) := \mathsf{redec}_{E\times A, E\times B}(\mathsf{lift}\ f) \quad \text{for} \quad f : \mathsf{Setoid}(\mathsf{Tri}A, \mathsf{eq}B) \ .$$

In Section 6 we generalize Example 33 – precomposition with a product – to more general relative comonads over suitable categories. However, this requires an axiomatization of the lift operation, more precisely of an important building block of it, the cut operation.

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

▶ **Definition 34** (Morphism of comodules)**.** Let M and $N : \mathcal{C} \to \mathcal{E}$ 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

$$\alpha_B \circ \mathsf{mcobind}^M(f) = \mathsf{mcobind}^N(f) \circ \alpha_A \ ,$$

in diagrammatic form

$$\begin{tikzcd} MA \arrow[r, "\mathsf{mcobind}^M(f)"] \arrow[d, "\alpha_A"'] & MB \arrow[d, "\alpha_B"] \\ NA \arrow[r, "\mathsf{mcobind}^N(f)"'] & NB. \end{tikzcd}$$

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 category $\mathcal{E}$.

The motivating examples for us to consider comodules and their morphisms are the following:

▶ **Example 35.** The family of destructors $\mathsf{tail}_A : \mathsf{Stream}A \to \mathsf{Stream}A$, indexed by a type A, is the carrier of a morphism of tautological comodules (over the relative comonad Stream),

$$\mathsf{tail} : \mathsf{Stream} \to \mathsf{Stream} \enspace .$$

▶ **Example 36.** The family of destructors $\mathsf{rest}_A : \mathsf{Tri}A \to \mathsf{Tri}(E \times A)$, indexed by a type A, of Example 10 is a morphism of comodules over the comonad Tri from the tautological comodule Tri to the comodule $\mathsf{Tri}(E \times \_)$ as defined in Example 33,

$$\mathsf{rest} : \mathsf{Tri} \to \mathsf{Tri}(E \times \_) \enspace .$$

▶ **Remark 37.** *The family of morphisms constituting a comodule morphism is actually natural with respect to the functoriality defined in Definition 31.*

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

▶ **Definition 38** (Pushforward comodule)**.** 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 $f : \mathcal{D}(SA, FB)$,

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

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

$$\tau_* : \mathsf{RComod}(T, \mathcal{E}) \to \mathsf{RComod}(S, \mathcal{E}) \enspace .$$

As presented in Definition 32, every relative comonad induces a comodule over itself. This extends to morphisms of relative comonads in the following sense:

▶ **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,

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

We conclude the section with some constructions of comodules:

▶ **Lemma 40.** *Let T be a comonad relative to $F : \mathcal{C} \to \mathcal{D}$, and let $M : \mathcal{C} \to \mathcal{E}$ be a comodule over T. Let $G : \mathcal{E} \to \mathcal{X}$ be a functor. Then $(G \circ M, G \circ \mathsf{mcobind}^M)$ is a comodule over T. This extends to morphisms of comodules: for N another comodule over T with target category $\mathcal{E}$, and $\alpha : M \to N$ a morphism of comodules, $G \circ \alpha, c \mapsto G(\alpha_c)$ is a morphism of comodules from $G \circ M$ to $G \circ N$. This yields a functor $\mathsf{RComod}(T, \mathcal{E}) \to \mathsf{RComod}(T, \mathcal{X})$.*

In particular, given any object $e : \mathcal{E}_0$, the constant functor $\mathcal{C} \to \mathcal{E}$ mapping to e is a comodule for any comonad relative to $F : \mathcal{C} \to \mathcal{D}$.

▶ **Lemma 41.** *Let T be a comonad relative to $F : \mathcal{C} \to \mathcal{D}$, and let $M, N : \mathcal{C} \to \mathcal{E}$ be comodules over T. Suppose that $\mathcal{E}$ has coproducts, denoted $e + e'$. Then $M + N, c \mapsto Mc + Nc$ is a comodule over T, with cosubstitution given by $\mathsf{mcobind}^M + \mathsf{mcobind}^N$. This construction extends to a coproduct on $\mathsf{RComod}(T, \mathcal{E})$.*

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.

This 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 42** (Category of models for Stream). 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 **morphism of models** $(S, t) \to (S', t')$ is given by a comonad morphism $\tau : S \to S'$ such that $\langle\tau\rangle \circ \tau_* t = t' \circ \langle\tau\rangle$, in diagrammatic form,

$$\begin{tikzcd} S \arrow[r, "\tau_* t"] \arrow[d, "\langle\tau\rangle"'] & S \arrow[d, "\langle\tau\rangle"] \\ S' \arrow[r, "t'"'] & S'. \end{tikzcd}$$

Note that the above diagram is a diagram in the category $\mathsf{RComod}(S', \mathsf{Setoid})$.

This defines a category of models of the signature of streams, with composition and identity inherited from those of comonad morphisms.

In the introduction we mentioned a more traditional notion of "model" for a signature of a coinductive data type, namely that of a coalgebra of some endofunctor. The relationship between models as in Definition 42 and coalgebras is as follows:

▶ **Remark 43** (Coalgebras for streams: forgetting cosubstitution). *Define the functor*

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

Then we have a forgetful functor from models for streams (in the sense of Definition 42) to coalgebras of F which, intuitively, forgets the cosubstitution structure. Indeed, given a model (S, t), the comonad S is, in particular, a functor $S : \mathsf{Type} \to \mathsf{Setoid}$ which constitutes the carrier of a coalgebra of F: the coalgebra structure map $S \to F(S) := \mathsf{eq} \times S$, as a map into a product, is assembled from the counit of S as the first component, and the comodule morphism $t : S \to S$, which is a natural transformation $t : S \to S$ as the second component.

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 defined above.

The category of models for Stream has a terminal object:

▶ **Theorem 44.** *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 42.*

More precisely, Theorem 44 says that the rules given in Figure 2 allow to prove that the category of models defined in Definition 42 has a terminal object. The proof of Theorem 44 being essentially the same as that of Theorem 56, we omit the former. However, a mechanized proof can be found in our Coq library, see Section 7.

The property of being terminal can be used to specify a map into streams, by equipping some comonad S relative to $\mathsf{eq} : \mathsf{Type} \to \mathsf{Setoid}$ with the structure of a model for streams, that is, with a comodule morphism $S \to S$. We do so for the comonad Tri:

▶ **Example 45.** 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 the composition

$$t^{\mathsf{diag}} := \mathsf{cut} \circ \mathsf{rest} : \mathsf{Tri} \to \mathsf{Tri} \ .$$

By terminality we obtain a morphism of models

$$(\mathsf{Tri}, t^{\mathsf{diag}}) \to (\mathsf{Stream}, \mathsf{tail})$$

which has, as underlying morphism of relative comonads, the one defined in Example 27. Compatibility of this map with cosubstitution in Tri and Stream, respectively, is for free.

▶ **Remark 46.** *Fix a type B. A result analogous to Theorem 44 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 like to 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 does not obtain the right auxiliary function cut for what is supposed to be the *terminal* such model, the pair $(\mathsf{Tri}, \mathsf{rest})$, where cut is used to define the comodule $\mathsf{Tri}(E \times \_)$. 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 47.** 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

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

satisfying the axioms

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

that is,

$$\begin{tikzcd} T(E\times A) \arrow[r, shift left, "T(\mathsf{pr}_2(E,A))"] \arrow[r, shift right, "\mathsf{cut}_A"'] & TA \arrow[r, "\mathsf{counit}_A"] & FA \end{tikzcd} \quad \text{and} \quad \begin{tikzcd} T(E\times A) \arrow[r, "\mathsf{cobind}(\mathsf{lift}\ f)"] \arrow[d, "\mathsf{cut}_A"'] & T(E\times B) \arrow[d, "\mathsf{cut}_B"] \\ TA \arrow[r, "\mathsf{cobind}(f)"'] & TB \end{tikzcd}$$

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

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

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

▶ **Definition 48.** Let (T, cut^T) and (S, cut^S) be two comonads relative to a functor F with cut relative to E as in Definition 47. A **morphism of comonads with cut** is a comonad morphism τ between the underlying comonads as in Definition 23 that commutes suitably with the respective cut operations, i.e. for any $A : \mathcal{C}_0$, $\mathsf{cut}^S_A \circ \tau_{E\times A} = \tau_A \circ \mathsf{cut}^T_A$:

$$\begin{tikzcd} T(E\times A) \arrow[r, "\mathsf{cut}^T_A"] \arrow[d, "\tau_{E\times A}"'] & TA \arrow[d, "\tau_A"] \\ S(E\times A) \arrow[r, "\mathsf{cut}^S_A"'] & SA. \end{tikzcd}$$

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 49** (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 47 by setting $\mathsf{ccut}_A := \mathsf{cut}_A := T\big(\mathsf{pr}_2(E, A)\big)$. (The extra "c" of* ccut *stands for "canonical".) It follows from the axioms of comonad morphisms that a comonad morphism $\tau : T \to S$ satisfies the equation of Definition 48 for the operations ccut^T and ccut^S thus defined. The morphism τ 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 50.** The relative comonad Tri from Example 20, together with the cut operation defined in Example 10, is a comonad with cut as in Definition 47.

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 51.** 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 47. 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 induced by that of M by

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

where the lift operation is the one defined in Definition 47.

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 52** (Pushforward commutes with product in context)**.** *The constructions of Definitions 51 and 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 48 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 52):

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

The construction of Definition 51 yields a categorical characterization of the rest destructor – more precisely, of its behavior with respect to cosubstitution as in Equation 1 – via the notion of comodule morphism:

▶ **Example 53.** Definition 51 is an axiomatization of Example 36: the relative comonad Tri is a relative comonad with cut, and the family of destructors $\mathsf{rest}_A : \mathsf{Tri}A \to \mathsf{Tri}(E \times A)$ constitutes a morphism of comodules

$$\mathsf{rest} : \mathsf{Tri} \to \mathsf{Tri}(E \times \_)$$

from the tautological comodule Tri to the composed comodule $\mathsf{Tri}(E \times \_)$ (obtained by precomposing the tautological comodule Tri with "product with E" as defined in Definition 51).

The axiomatization of the cut operation now allows us to define models for the signature of infinite triangular matrices:

▶ **Definition 54** (Models for infinite triangular matrices)**.** Let $E : \mathsf{Type}$ be a fixed type. Let $\mathcal{T} = \mathcal{T}_E$ be the category of **models for infinite triangular matrices** where an object is a pair (T, rest^T) consisting of

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

such that for any set A, $\mathsf{rest}^T_A \circ \mathsf{cut}^T_A = \mathsf{cut}^T_{E\times A} \circ \mathsf{rest}^T_{E\times A}$.

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

$$\begin{tikzcd}
T(E \times \_) \arrow[r, "\mathsf{rest}^T(E\times\_)"] \arrow[d, "\mathsf{cut}^T"'] & T(E \times E \times \_) \arrow[d, "\mathsf{cut}^T(E\times\_)"] \\
T \arrow[r, "\mathsf{rest}^T"'] & T(E \times \_).
\end{tikzcd}$$

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,

$$\begin{tikzcd}
\tau_* T \arrow[r, "\tau_*(\mathsf{rest}^T)"] \arrow[d, "\langle\tau\rangle"'] & \tau_* T(E \times \_) \arrow[d, "\langle\tau\rangle(E\times\_)"] \\
S \arrow[r, "\mathsf{rest}^S"'] & S(E \times \_)\ .
\end{tikzcd} \tag{8}$$

Here we silently insert an isomorphism as in Remark 52 in the upper right corner.

Analogously to the signature for streams, there is a forgetful functor from models for triangular matrices to coalgebras of a specific higher-order endofunctor:

▶ **Remark 55** (Coalgebras for Tri)**.** *Let $E : \mathsf{Type}$ be a type. Define the functor*

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

There is 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, rest^T), 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 map on T, being a map $T \to \mathsf{eq} \times T(E \times \_)$ into a product, is assembled from the counit of T (to give the first component) and the comodule morphism $\mathsf{rest}^T : T \to T(E \times \_)$ (to give the second component). Note that both the counit and rest^T are natural transformations.*

▶ **Theorem 56.** *The pair* $(\mathsf{Tri}, \mathsf{rest})$ *consisting of the relative comonad with cut* Tri *of Example 50 together with the morphism of comodules* rest *of Example 36, constitutes the terminal model of triangular matrices.*

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

$$\mathsf{top} \circ \bigcirc := \mathsf{counit}^T \quad \text{and} \tag{9}$$
$$\mathsf{rest} \circ \bigcirc := \bigcirc \circ \mathsf{rest}^T\ . \tag{10}$$

```
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'.
```

Figure 8 Definition of Tri and bisimilarity using Coq's CoInductive vernacular.

Using the introduction axiom for bisimilarity we show that the map $\bigcirc$ commutes 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 (9) is forced upon us since we want $\bigcirc$ to constitute a morphism of comonads – the equation directly corresponds to one of the axioms. Equation (10) is forced upon us by the diagram of Equation (8), which a morphism of models has to make commute.

The same argument is used to show, again by use of the introduction rule for bisimilarity, that any two morphisms of models $\tau, \rho : (T, \mathsf{rest}^T) \to (\mathsf{Tri}, \mathsf{rest})$ are equal in the sense of being pointwise bisimilar, thus concluding the proof. ◀

7 Formalization in Coq

All our definitions and theorems are mechanized in the proof assistant Coq [10]. In the formalization, we axiomatize the considered coinductive type families as given by the rules of Figures 2 and 4. In order to ensure consistency, we

1. use Coq *module* types to encapsulate the axioms, implemented via the Axiom vernacular, and
2. instantiate each module type by defining streams and triangular matrices as coinductive types using the CoInductive vernacular, as shown in Figure 8 for the example of Tri.

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. Our results are hence axiom-free with respect to the theory implemented in the Coq proof assistant.

The Coq source files are available from the project web site [5]. The correspondences between formalized statements and the statements in this article are given in Table 1.

Prior to version 8.5 of Coq, a duplication of the definition of *setoids* was necessary in order to avoid a universe inconsistency: indeed, the type of setoids is both the target type of a field in the record of categories as well as the type of objects of the category of setoids, which leads to a complicated graph of universe constraints. In order to work around a universe inconsistency, the type of setoids to be used as objects of the category of setoids hence had to be (re)defined later to obtain a sufficiently large universe level.

Version 8.5 of Coq introduces a new, optional, universe polymorphism [21]. In our code repository [5], we provide a version of our code for Coq 8.4pl6 and two versions for Coq 8.5 (beta2 at the time of writing); one where universe polymorphism is not employed, and where hence the aforementioned duplication of code is necessary, and one where polymorphism is used to get rid of that duplication. However, in that latter version, getting our files accepted by Coq then required giving up on canonical structures, which do not seem to play well (yet) with universe polymorphism.

Table 1 Correspondence of informal and formal definitions.

Informal	Reference	Formal
Category		`Category`
Functor		`Functor`
Relative comonad	Def. 12	`RelativeComonad`
Streams as comonad	Ex. 17	`Stream`
Triangular matrices as comonad	Ex. 20	`Tri`
Comodule over comonad	Def. 30	`Comodule`
Tautological comodule (of T)	Def. 32	`tcomod`, notation `<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. 42	`Stream`
Stream is terminal	Thm. 44	`StreamTerminal.Terminality`
Relative comonad with cut	Def. 47	`RelativeComonadWithCut`
Precomposition with product	Def. 51	`precomposition_with_product`
Models for triangular matrices	Def. 54	`TriMat`
Tri is terminal	Thm. 56	`TriMatTerminal.Terminality`

8 Conclusion and future work

We have given a categorical semantics for some homogeneous (streams and trees) and heterogeneous (infinite triangular matrices) codata type families, using the notion of relative comonad and comodule over such comonads.

It remains to investigate more general forms of trees, in particular more general forms of heterogeneity than the one we have considered here. This requires a semantic analysis of the conditions that a functor (on the category of types) responsible for heterogeneity needs to satisfy in order to allow the lifting of a (co)substitution rule.

Acknowledgments. We are grateful to André Hirschowitz, Ralph Matthes and Paige North for many helpful discussions. We thank the referees for their suggestions and comments, and the editors for their efforts in organizing these proceedings.

References

1 Andreas Abel, Brigitte Pientka, David Thibodeau, and Anton Setzer. Copatterns: programming infinite structures by observations. In Roberto Giacobazzi and Radhia Cousot, editors, *Principles of Programming Languages*, pages 27–38. ACM, 2013. doi:10.1145/2429069.2429075.

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

3 Benedikt Ahrens. Modules over relative monads for syntax and semantics. *Mathematical Structures in Computer Science*, FirstView:1–35, 2014. doi:10.1017/S0960129514000103.

4 Benedikt Ahrens, Paolo Capriotti, and Régis Spadotti. Non-Wellfounded Trees in Homotopy Type Theory. In Thorsten Altenkirch, editor, *13th International Conference on Typed*

Lambda Calculi and Applications (TLCA 2015), volume 38 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 17–30, Dagstuhl, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. `https://hott.github.io/M-types/`.

5 Benedikt Ahrens and Régis Spadotti. Terminal semantics for codata types in intensional Martin-Löf type theory: Coq code. `http://benediktahrens.github.io/coinductives/`.

6 Thorsten Altenkirch, James Chapman, and Tarmo Uustalu. Monads need not be endofunctors. In C.-H. Luke Ong, editor, *Foundations of Software Science and Computational Structures*, volume 6014 of *LNCS*, pages 297–311. Springer, 2010.

7 Thorsten Altenkirch, Neil Ghani, Peter Hancock, Conor McBride, and Peter Morris. Indexed Containers. Unpublished version, `http://www.cs.nott.ac.uk/~txa/publ/jcont.pdf`.

8 Thorsten Altenkirch and Bernhard Reus. Monadic presentations of lambda terms using generalized inductive types. In Jörg Flum and Mario Rodríguez-Artalejo, editors, *Proceedings of Computer Science Logic*, volume 1683 of *Lecture Notes in Computer Science*, pages 453–468. Springer, 1999. doi:10.1007/3-540-48168-0_32.

9 Steve Awodey, Nicola Gambino, and Kristina Sojakova. Inductive Types in Homotopy Type Theory. In *Proceedings of Symposium on Logic in Computer Science*, pages 95–104. IEEE, 2012. doi:10.1109/LICS.2012.21.

10 The Coq Development Team. The Coq Proof Assistant. `http://coq.inria.fr`, 2015.

11 Thierry Coquand. Infinite objects in type theory. In Henk Barendregt and Tobias Nipkow, editors, *Types for Proofs and Programs 1993*, volume 806 of *Lecture Notes in Computer Science*, pages 62–78. Springer, 1993. doi:10.1007/3-540-58085-9_72.

12 Peter Dybjer. Representing Inductively Defined Sets by Wellorderings in Martin-Löf's Type Theory. *Theor. Comput. Sci.*, 176(1-2):329–335, 1997. doi:10.1016/S0304-3975(96)00145-4.

13 Marcelo Fiore, Gordon Plotkin, and Daniele Turi. Abstract syntax and variable binding. In *Proceedings of the Symposium on Logic in Computer Science*, pages 193–202, Washington, DC, USA, 1999. IEEE Computer Society. doi:10.1109/LICS.1999.782615.

14 André Hirschowitz and Marco Maggesi. Modules over monads and initial semantics. *Inf. Comput.*, 208(5):545–564, 2010. doi:10.1016/j.ic.2009.07.003.

15 Gérard P. Huet and Amokrane Saïbi. Constructive category theory. In Gordon D. Plotkin, Colin Stirling, and Mads Tofte, editors, *Proof, Language, and Interaction, Essays in Honour of Robin Milner*, pages 239–276. The MIT Press, 2000.

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

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 Nils Anders Danielsson and Bengt Nordström, editors, *Workshop on Types for Proofs and Programs*, volume 19 of *LIPIcs*, pages 55–69. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2011. doi:10.4230/LIPIcs.TYPES.2011.55.

19 Ieke Moerdijk and Erik Palmgren. Wellfounded trees in categories. *Ann. Pure Appl. Logic*, 104(1-3):189–218, 2000. doi:10.1016/S0168-0072(00)00012-9.

20 John Power. Abstract syntax: Substitution and binders. *Electron. Notes Theor. Comput. Sci.*, 173:3–16, 4 2007. doi:10.1016/j.entcs.2007.02.024.

21 Matthieu Sozeau and Nicolas Tabareau. Universe polymorphism in coq. In Gerwin Klein and Ruben Gamboa, editors, *Proceedings of Interactive Theorem Proving*, volume 8558 of *Lecture Notes in Computer Science*, pages 499–514. Springer, 2014.

22 Miki Tanaka and John Power. A unified category-theoretic formulation of typed binding signatures. In *Proceedings of the workshop on Mechanized reasoning about languages with variable binding*, MERLIN'05, pages 13–24. ACM, 2005. doi:10.1145/1088454.1088457.

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

24 Tarmo Uustalu and Varmo Vene. The dual of substitution is redecoration. In Kevin Hammond and Sharon Curtis, editors, *Scottish Functional Programming Workshop*, volume 3 of *Trends in Functional Programming*, pages 99–110. Intellect, 2001.