Greatest Bisimulations for Binary Methods

Hendrik Tews [1]

TU Dresden, Institut für Theoretische Informatik,
D-01062 Dresden, Germany.

Abstract

A generalised notion of coalgebra that is capable of modelling binary methods as they occur in object-oriented programming has been introduced in [14]. An important problem with this generalisation is that bisimulations are not closed under union and that a greatest bisimulation does not exists in general. There are two possible approaches to improve this situation: First, to strengthen the definition of bisimulation, and second, to place constraints on the coalgebras (i.e., on the behaviour of the binary methods). In this paper I combine both approaches to show that (under reasonable assumptions) the greatest bisimulation does exist for all coalgebras of extended polynomial functors.

1 Introduction

The term *binary method* stems from object-oriented programming. A method is called a binary method if it takes an additional second argument of its hosting class. The canonical example is the method

$$\mathsf{equal}: \qquad \mathsf{Self} \times \mathsf{Self} \longrightarrow \mathsf{bool}$$

Here, Self stands for the type of the current class. In a typical object-oriented language the first argument of type Self is implicit and does not show up in the interface of the method equal. Binary methods are famous for posing problems in theoretical accounts on object-orientation, see [2].

A promising approach to give a semantics for object-orientation is based on coalgebras [11]. A coalgebra is simply a function with a structured codomain like

$$\mathsf{next\_state}: \qquad \mathsf{Self} \longrightarrow \mathsf{Self} \times \mathbb{N} + \mathbf{1}$$

The codomain of the coalgebra describes possible observations and successor states. It is typically described with an endofunctor. For the preceding example one would

[1] Email: tews@tcs.inf.tu-dresden.de

This is a preliminary version. The final version will be published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

use $K(X) = (X \times \mathbb{N}) + \mathbf{1}$. The advantage of a coalgebraic approach to object-orientation is that it directly addresses terms like behaviour, behavioural indistinguishability (bisimulation), and information hiding. It further supports coinduction as a definition and proof principle and modal logics via the notion of invariants, see for instance [7,12].

However, the use of coalgebras for endofunctors to model classes (as suggested in [11]) does *not* cover binary methods. The problem is that endofunctors are not sufficient to model class signatures with binary methods. The easiest solution to solve this problem is to treat binary methods as definitional extensions [6]. This way binary methods are not considered as part of the signature, but as external functions which are uniformly defined in terms of the operations in the signature.

Hennicker and Kurz suggest in [3] to formalise binary methods as algebraic extensions to coalgebraic signatures. This approach works well for those binary methods that have a codomain of Self (so the canonical example equal is excluded) and whose behaviour is completely determined by the operations in the coalgebraic signature.

A rigorous solution that allows to model methods of arbitrary types, including binary methods, has been proposed by the present author in [14]. The idea is to use bivariant functors $\mathbf{Set}^{\mathrm{op}} \times \mathbf{Set} \longrightarrow \mathbf{Set}$ to model signatures and to work with appropriately generalised notions of coalgebra and bisimulation. If H is such a bivariant functor then a H–coalgebra is a function $X \longrightarrow H(X, X)$. This allows to model exotic method types like [2]

$$\mathsf{Self} \times (\mathbb{N} \Rightarrow \mathsf{Self} + \mathbf{1}) \longrightarrow \boxed{\cdots \ \mathsf{Self} \ \cdots} \qquad (*)$$

Here one would use a functor $H(Y, X) = (\mathbb{N} \Rightarrow (Y + \mathbf{1})) \Rightarrow \cdots$.

One problem with the approach of [14] is that generalised coalgebras are not as well-behaved as coalgebras for weak-pullback preserving endofunctors. A careful investigation in [14] yields the class of *extended polynomial functors* (see page 6 for a definition) as a reasonable compromise between expressivity and general structural properties. Extended polynomial functors cover the preceding example $(*)$. For coalgebras of extended polynomial functors one can show that bisimulations and invariants are closed under intersection, coalgebra morphisms are functional bisimulations, and many other standard results (again, I refer to [14]). An important question left open in [14] is under which conditions bisimilarity as greatest bisimulation does exists for generalised coalgebras.

This omission stimulated the interest of Poll and Zwanenburg. In [10] they define *dialgebras* as collections $(F_i^I(X) \longrightarrow F_i^O(X))$, where X is the state space of the dialgebra and F_i^I and F_i^O are functors $\mathbf{Set} \longrightarrow \mathbf{Set}$ describing the input and the output type of the i–th operation in the dialgebra. Poll and Zwanenburg restrict the functors that may occur in their dialgebras to those functors that can be built up from constants, the identity, products, and coproducts. With these assumptions they prove in [10] that for dialgebras a greatest bisimulation equivalence does always

[2] I use $X \Rightarrow Y$ to denote the function space (exponent) between X and Y.

exist. This result implies that for some (proper) extended polynomial functors a greatest bisimulation does exist for all coalgebras. However, the result of Poll and Zwanenburg does not apply to all extended polynomial functors, because not every extended polynomial functor can be represented as a dialgebra.

The work of Poll and Zwanenburg spurred me on, and based on their ideas the present paper makes the following contributions: Section 4 generalises the cited result of Poll and Zwanenburg slightly and adapts it to the framework of [14]. The main result in Section 4 is Theorem 4.6. It says that for *extended cartesian functors* (a proper subclass of extended polynomial functors) those bisimulations that are partial equivalence relations form a complete lattice.

Example 4.7 shows that Theorem 4.6 is sharp: It contains a coalgebra for an extended polynomial functor which has an infinitely ascending chain of bisimulations without an upper bound. The Example 4.7 is based on an undecidable function. This raises the question if Theorem 4.6 can be generalised by placing certain restrictions on the coalgebras. Section 5 presents such a restriction: *finitely based coalgebras*. It enables the proof of Theorem 5.7 that says (roughly) that a greatest bisimulation does exist for those coalgebras of extended polynomial functors that are finitely based.

The contents of the other sections is as follows: The following Section 2 introduces some nonstandard notation. Section 3 copies the important definitions from [14] to make this paper self contained. In particular, Section 3 defines various classes of functors and the notion of coalgebra and bisimulations. In comparison with [14] there is a small but important difference: In this paper all functors can contain the functor List : $\mathbf{Set} \longrightarrow \mathbf{Set}$ as an ingredient functor. (The functor List builds the set of (finite) lists over its argument.)

A special feature of the present paper is that all results and examples have been —to the extend that is feasible— formalised and proved in the theorem prover PVS [9]. The source code is available in the world wide web at URL wwwtcs.inf.tu–dresden.de/~tews/binary/.

In the remainder of the introduction I discuss why one should care at all about methods like $(*)$ above. For that purpose consider a coalgebraic signature of one place buffers with two operations

$$\begin{array}{ll} \mathsf{store}: & \mathsf{Self} \times I \longrightarrow \mathsf{Self} \\ \mathsf{fetch}: & \mathsf{Self} \longrightarrow I + \mathbf{1} \end{array}$$

One would add an axiom like $\mathsf{fetch}(\mathsf{store}(x, i)) = \kappa_1\, i$ and allow fetch to return $* \in \mathbf{1}$ if nothing has been stored before.

Let us now model *chained* buffers, such that a buffer b_1 (possibly) contains a reference to a buffer b_2. Via this reference methods operating on buffer b_1 can invoke methods that operate on b_2. An example of such a method is the operation push. When push is invoked on buffer b_1 it stores the contents of b_1 in the buffer b_2 by calling the method store on b_2.

The chained buffers shall model side effects in the following sense: If a third buffer b_3 (also) contains a reference to b_2, then the state change in b_2 when pushing in b_1 should be visible in b_3. To obtain this effect in a purely functional environment (like a theorem prover) one has to pass the current environment of buffers into the method push. Such an environment of buffers is a function $\mathbb{N} \longrightarrow \mathsf{Self} + \mathbf{1}$. The type of push is then [3]

$$\mathsf{push}: \quad \mathsf{Self} \times (\mathbb{N} \Rightarrow \mathsf{Self} + \mathbf{1}) \longrightarrow (\mathbb{N} \Rightarrow \mathsf{Self} + \mathbf{1})$$

This method push requires extended polynomial functors.

2 Preliminaries

This section introduces some standard and some nonstandard notation for this paper. In the whole paper I work in the category $\mathbf{Set}$ of sets and total functions. I use $\times$ to denote the Cartesian product with the projections $X \xleftarrow{\pi_1} X \times Y \xrightarrow{\pi_2} Y$. The disjoint union (the coproduct) is denoted by $+$ together with the injections $X \xrightarrow{\kappa_1} X + Y \xleftarrow{\kappa_2} Y$. I write $X \Rightarrow Y$ for the function space (exponent) between X and Y. The composition of two functions $f : X \longrightarrow Y$ and $g : Y \longrightarrow Z$ is $g \circ f : X \longrightarrow Z$.

A binary relation between X and Y is written as $R \subseteq X \times Y = \{(x : X, y : Y) \mid R(x, y)\}$. The equality relation on an arbitrary set A is $\mathrm{Eq}(A) = \{(a, a) \mid a \in A\}$. The union over an I–indexed set of relations $(R_i)_{i \in I}$ is given as $\bigcup_i R_i = \{(x, y) \mid \exists i \,.\, x \, R_i \, y\}$. For a relation $R \subseteq X \times X$ its *domain* is a predicate on X, defined as $\mathsf{dom}(R) = \{x \mid \exists x' \,.\, x \, R \, x' \vee x' \, R \, x\}$. A relation is *partially reflexive* if it is reflexive on its domain, that is if $x \, R \, y$ implies $x \, R \, x$ and $y \, R \, y$. The relation R is a *partial equivalence relation* if it is symmetric and transitive. The least partial equivalence relation containing R is denoted with $\overline{R}$. Taking the least partial equivalence relation is a closure operation, in particular $\overline{\overline{R}} = \overline{R}$ and $R \subseteq S$ implies $\overline{R} \subseteq S$ if S is a partial equivalence relation. A *zigzag* in a relation R is a finite sequence $x_1, \ldots, x_n$ such that $x_i \, R \, x_{i+1}$ or $x_{i+1} \, R \, x_i$ holds for all $i < n$. The closure $\overline{R}$ can be described as follows: $x \, \overline{R} \, y$ holds if and only if there is a zigzag $x, \ldots, y$ in R.

The cartesian closed structure in $\mathbf{Set}$ can be lifted to relations as follows. Let $S \subseteq U \times V$ and $R \subseteq X \times Y$ be relations:

$$\begin{aligned}
S \times_{\mathrm{R}} R \;&\subseteq\; (U \times X) \times (V \times Y) &&= \big\{((u, x), (v, y)) \mid S(u, v) \;\wedge\; R(x, y)\big\} \\
S +_{\mathrm{R}} R \;&\subseteq\; (U + X) \times (V + Y) &&= \big\{(\kappa_1 \, u, \kappa_1 \, v) \mid S(u, v)\big\} \\
& && \quad \cup \;\; \big\{(\kappa_2 \, x, \kappa_2 \, y) \mid R(x, y)\big\} \\
S \Rightarrow_{\mathrm{R}} R \;&\subseteq\; (U \Rightarrow X) \times (V \Rightarrow Y) &&= \big\{(f, g) \mid \forall u : U, v : V \,.\, S(u, v) \\
& && \qquad \text{implies} \quad R(f(u), g(v))\big\}
\end{aligned}$$

[3] To keep the example within reasonable size I deliberately neglect some important issues: There should be methods to chain buffers, to assign addresses to buffers, and so on.

In a suitable setting (see [8]) the operations $\times_{\mathrm{R}}, +_{\mathrm{R}}$, and $\Rightarrow_{\mathrm{R}}$ can be extended to fibred functors over the cartesian closed structure of $\mathbf{Set}$. This fibred structure is not important for this paper. However, I do use that $\times_{\mathrm{R}}$ and $+_{\mathrm{R}}$ are monotone (with respect to inclusion) in both arguments and that $\Rightarrow_{\mathrm{R}}$ is antimonotone in the first and monotone in the second argument.

In the following I use the functor $\mathsf{List} : \mathbf{Set} \longrightarrow \mathbf{Set}$ that maps a set A to the set of finite lists over A. Formally, $\mathsf{List}(A)$ is obtained as (the carrier of) the initial algebra of the functor $F^A_{\mathsf{List}}(X) = A \times X + \mathbf{1}$. I use $\mathsf{nil} : \mathsf{List}(A)$ and $\mathsf{cons} : A \times \mathsf{List}(A) \longrightarrow \mathsf{List}(A)$ to denote the initial algebra on $\mathsf{List}(A)$.

The functor List can also be lifted to relations. The general theory is described in [4]. For a relation $R \subseteq X \times Y$ the lifting $\mathrm{RelList}(R) \subseteq \mathsf{List}(X) \times \mathsf{List}(Y)$ is characterised as follows:

$$\mathsf{RelList}(R)(l_1, l_2) \quad = \quad \begin{cases} \mathsf{true} & \text{if } l_1 = \mathsf{nil} \;\wedge\; l_2 = \mathsf{nil} \\ R(x,y) \;\wedge\; \mathsf{RelList}(R)(l_1', l_2') & \text{if } l_1 = \mathsf{cons}(x, l_1') \\ & \quad \wedge\, l_2 = \mathsf{cons}(y, l_2') \\ \mathsf{false} & \text{otherwise} \end{cases}$$

The following lemma describes the interplay between the lifted operations and taking unions of relations.

Lemma 2.1 *Let $(S_i \subseteq U \times V)_{i \in I}$ and $(R_i \subseteq X \times Y)_{i \in I}$ be arbitrary I–indexed collections of relations. Then*

$$\begin{array}{rcl} \bigcup_i \mathrm{RelList}(R_i) & \subseteq & \mathrm{RelList}\big(\bigcup_i R_i\big) \\ \bigcup_i (S_i \times_{\mathrm{R}} R_i) & \subseteq & \bigcup_i S_i \;\times_{\mathrm{R}}\; \bigcup_i R_i \\ \bigcup_i (S_i +_{\mathrm{R}} R_i) & = & \bigcup_i S_i \;+_{\mathrm{R}}\; \bigcup_i R_i \\ \bigcup_i (\mathrm{Eq}(A) \Rightarrow_{\mathrm{R}} R_i) & \subseteq & \mathrm{Eq}(A) \;\Rightarrow_{\mathrm{R}}\; \bigcup_i R_1 \end{array} \qquad \square$$

It is easy to find examples in which the subset relations in the preceding lemma are strict. Moreover, the two expressions for the general exponent $\bigcup_i (S_i \Rightarrow_{\mathrm{R}} R_i)$ and $\bigcup_i S_1 \;\Rightarrow_{\mathrm{R}}\; \bigcup_i R_1$ are unrelated in general.

3 Coalgebras for Generalised Polynomial Functors

In this paper I consider different classes of *polynomial* functors on the category $\mathbf{Set}$ of sets and total functions. These functors are called polynomial because they are inductively build up from identity, constants, (co–)products, and the exponent. I distinguish five classes of polynomial functors. The only difference between these classes is the kind of exponent that is allowed. In the following A stands for an arbitrary constant set.

Cartesian Functors:

$$K(X) \;=\; A \;\mid\; X \;\mid\; \mathsf{List}(K_1(X)) \;\mid\; K_1(X) \times K_2(X) \;\mid\; K_1(X) + K_2(X)$$

Class of Functors	Property	Example
cartesian	no arguments	$\mathsf{Self} \longrightarrow \boxed{\cdots}$
polynomial	constant arguments	$\mathsf{Self} \times A \longrightarrow \boxed{\cdots}$
extended cartesian	no functional arguments	$\mathsf{Self} \times \mathsf{Self} \times \mathsf{Self} \longrightarrow \boxed{\cdots}$ $\mathsf{Self} \times (\mathsf{Self} + A) \longrightarrow \boxed{\cdots}$
extended polynomial	functional arguments with constant domain	$\mathsf{Self} \times (A \Rightarrow \mathsf{Self}) \longrightarrow \boxed{\cdots}$
higher-order polynomial	arbitrary arguments	$\mathsf{Self} \times (\mathsf{Self} \Rightarrow A) \longrightarrow \boxed{\cdots}$ $\mathsf{Self} \times (\mathsf{Self} \Rightarrow \mathsf{Self}) \longrightarrow \boxed{\cdots}$

Table 1
Classes of functors in this paper

Polynomial functors:

$$\begin{aligned} F(X) \;=\; & A \;\mid\; X \;\mid\; \mathsf{List}(F_1(X)) \;\mid\; F_1(X) \times F_2(X) \;\mid \\ & F_1(X) + F_2(X) \;\mid\; A \Rightarrow F_1(X) \end{aligned}$$

Extended Cartesian Functors:

$$\begin{aligned} H^K(Y,X) \;=\; & A \;\mid\; X \;\mid\; \mathsf{List}(H_1^K(Y,X)) \;\mid\; H_1^K(Y,X) \times H_2^K(Y,X) \;\mid \\ & H_1^K(Y,X) + H_2^K(Y,X) \;\mid\; K(Y) \Rightarrow H_1^K(Y,X) \end{aligned}$$

Extended Polynomial Functors:

$$\begin{aligned} H^F(Y,X) \;=\; & A \;\mid\; X \;\mid\; \mathsf{List}(H_1^F(Y,X)) \;\mid\; H_1^F(Y,X) \times H_2^F(Y,X) \;\mid \\ & H_1^F(Y,X) + H_2^F(Y,X) \;\mid\; F(Y) \Rightarrow H_1^F(Y,X) \end{aligned}$$

Higher-order Polynomial Functors:

$$\begin{aligned} H(Y,X) \;=\; & A \;\mid\; X \;\mid\; \mathsf{List}(H_1(Y,X)) \;\mid\; H_1(Y,X) \times H_2(Y,X) \;\mid \\ & H_1(Y,X) + H_2(Y,X) \;\mid\; H_1(X,Y) \Rightarrow H_2(Y,X) \end{aligned}$$

Cartesian Functors and polynomial functors are endofunctors $\mathbf{Set} \longrightarrow \mathbf{Set}$, so they take only one argument. The other classes describe bivariant functors with a contravariant first and a covariant second argument. When used to model coalgebraic signatures the above classes place different restrictions on the method types that can be modelled. See Table 1 for examples.

Higher-order polynomial functors are the most general class defined above. I do not investigate higher-order polynomial functors in this paper. However, I use them to give definitions that apply to all considered classes of functors. Extended polynomial functors restrict the domain in the clause for the exponent to polynomial functors. Extended cartesian functors require a cartesian functor for the domain. In comparison with [14] this paper explicitly allows the functor List as ingredient functor. All results of [14] apply also to the more general classes of functors of this paper. The necessary proofs are in the PVS formalisation.

For a comparison with the results of Poll and Zwanenburg in [10] note that every dialgebra with state space X corresponds to a pair $\langle c, a\rangle$, where c is a coalgebra for some extended cartesian functor and a is a constant in $F(X)$ for some polynomial functor F. There are coalgebras for extended cartesian functors that cannot be represented as a dialgebra. For example the coalgebras for the functor $G(Y, X) = (Y \Rightarrow A) + (X \times A)$.

Definition 3.1 Let $H : \mathbf{Set}^{\mathrm{op}} \times \mathbf{Set} \longrightarrow \mathbf{Set}$ be a higher-order polynomial functor. A *H–coalgebra* is a function $c : X \longrightarrow H(X, X)$. Let $d : Y \longrightarrow H(Y, Y)$ be another H–coalgebra. A function $f : X \longrightarrow Y$ is an *H–coalgebra morphism* $f : c \longrightarrow d$, if the following diagram commutes:

$$
\begin{tikzcd}
X \arrow[r, "c"] \arrow[dd, "f"'] & H(X,X) \arrow[dr, "H(X,f)"] & \\
& & H(X,Y) \\
Y \arrow[r, "d"] & H(Y,Y) \arrow[ur, "H(f,Y)"'] &
\end{tikzcd}
$$

As a functor H preserves composition and identities, therefore the above defines the category $CoAlg(H)$ of H–coalgebras (in $\mathbf{Set}$) for each higher-order polynomial functor H.

For polynomial and cartesian functors the preceding pentagon collapses to the familiar square.

Example 3.2 To model the signature of chained buffers from the introduction one needs the extended polynomial functor

$$G_{\mathsf{Buf}}(Y, X) \quad = \quad (I \Rightarrow X) \times (I + \mathbf{1}) \times \big((\mathbb{N} \;\Rightarrow\; Y + \mathbf{1}) \;\Rightarrow\; \mathbb{N} \;\Rightarrow\; X + \mathbf{1}\big)$$

Instead of functions one can use (association) lists over $\mathbb{N} \times$ Self to model the environment. Then the type of push changes and in order to model the signature one needs the extended cartesian functor

$$G'_{\mathsf{Buf}}(Y, X) \quad = \quad (I \Rightarrow X) \times (I + \mathbf{1}) \times \big(\mathsf{List}(\mathbb{N} \times Y) \;\Rightarrow\; \mathsf{List}(\mathbb{N} \times X)\big) \quad \blacksquare$$

There are two approaches in the literature to define bisimulations: Either in the Aczel/Mendler style [1] or by exploiting relation lifting as suggested by Hermida and Jacobs [5]. For the scope of this paper the choice in defining bisimulations is mainly a matter of taste, because for extended polynomial functors both approaches yield identical notions of bisimulation [14]. I prefer to follow the Hermida/Jacobs approach.

Definition 3.3 Let $S \subseteq V \times Y$ and $R \subseteq U \times X$ be two relations, and H a higher-order polynomial functor. Its *relation lifting* $\mathrm{Rel}(H)(S,R) \subseteq H(V,U) \times H(Y,X)$ is defined by induction on the structure of H (see page 4f for the description of the operators $\times_{\mathrm{R}}, +_{\mathrm{R}}, \Rightarrow_{\mathrm{R}}$, and RelList):

$$
\begin{array}{llcll}
H = A: & \mathrm{Rel}(H)(S,R) & = & \mathsf{Eq}(A) & = \{(a,a) \mid a \in A\} \\
H = X: & \mathrm{Rel}(H)(S,R) & = & R & \\
H = \mathsf{List}(H_1): & \mathrm{Rel}(H)(S,R) & = & \mathrm{RelList}(\mathrm{Rel}(H_1)(S,R)) & \\
H = H_1 \times H_2: & \mathrm{Rel}(H)(S,R) & = & \mathrm{Rel}(H_1)(S,R) \times_{\mathrm{R}} \mathrm{Rel}(H_2)(S,R) & \\
H = H_1 + H_2: & \mathrm{Rel}(H)(S,R) & = & \mathrm{Rel}(H_1)(S,R) +_{\mathrm{R}} \mathrm{Rel}(H_2)(S,R) & \\
H = H_1 \Rightarrow H_2: & \mathrm{Rel}(H)(S,R) & = & \mathrm{Rel}(H_1)(R,S) \Rightarrow_{\mathrm{R}} \mathrm{Rel}(H_2)(S,R) &
\end{array}
$$

In the preceding definition the relation lifting for higher-order polynomial functors takes a contravariant and covariant argument relation. For cartesian and polynomial functors the contravariant argument is never used. For these functors I drop the contravariant argument and write $\mathrm{Rel}(F)(R)$ for their relation lifting.

Definition 3.4 Let $c : X \longrightarrow H(X,X)$ and $d : Y \longrightarrow H(Y,Y)$ be two coalgebras for a higher-order polynomial functor H. A relation $R \subseteq X \times Y$ is a H–*bisimulation* for c and d if for all $x \in X, y \in Y$

$$R(x,y) \quad \text{implies} \quad \mathrm{Rel}(H)(R,R)(c(x),d(y))$$

In the present paper I only consider bisimulations $R \subseteq X \times X$ on one coalgebra. Such a bisimulation R is called a *(partial) bisimulation equivalence* if R is a (partial) equivalence relation. Note that the preceding definition does not require such bisimulations to be reflexive or transitive (as one would expect for a decent notion of behavioural equality). For coalgebras of polynomial functors this is not a problem, because there one can prove that every bisimulation is contained in some bisimulation equivalence [13]. However, this fails as soon as binary methods are present.

Example 3.5 This example shows two bisimulations such that their union is not a bisimulation. Consider the extended cartesian functor $G(Y,X) = Y \Rightarrow \mathsf{bool}$, which corresponds to a coalgebraic signature with just one method equal. Take the set $A = \{a_1, a_2\}$ as state space and define a G-coalgebra c as follows.

$$c(x)(y) \quad = \quad \mathsf{if}\ x = y\ \mathsf{then}\ \mathsf{true}\ \mathsf{else}\ \mathsf{false}\ \mathsf{endif}$$

Then the relation $R \stackrel{\text{def}}{=} \{(a_1,a_2),(a_2,a_1)\}$ is a bisimulation for c and so is $\mathrm{Eq}(A)$.

However, their union is not a bisimulation. The problem here is that the informal reasoning along the lines “if a_1 is behaviourally indistinguishable from a_2 then $c(a_1)(a_1)$ is indistinguishable from $c(a_1)(a_2)$” fails for the bisimulation R because R does not contain the pair (a_1, a_1). ■

4 Greatest Bisimulation Equivalences

Example 3.5 shows that the straightforward generalisation in Definition 3.4 yields a notion of bisimulation that is too weak. An obvious solution is to consider only bisimulations with additional properties. For instance Proposition 4.4 in [14] proves that reflexive bisimulations form a (incomplete) lattice for coalgebras of extended polynomial functors.

In this section I consider partial bisimulation equivalences. They make it possible to adapt the result about the existence of a greatest bisimulation equivalences of Poll and Zwanenburg in [10]. All the following lemmas are towards Theorem 4.6. Their interplay becomes clear in the proof of Lemma 4.5. I start the development with three simple observations that deal with side conditions below.

Lemma 4.1

(i) *Relation lifting for polynomial functors preserves partially reflexive relations. That is, if* $R \subseteq X \times X$ *is partially reflexive then so is* $\mathrm{Rel}(F)(R)$ *for all polynomial functors* F*.*

(ii) *Relation lifting for higher-order polynomial functors preserves partial equivalence relations. That is, if* $S \subseteq Y \times Y$ *and* $R \subseteq X \times X$ *are partial equivalence relations then so is* $\mathrm{Rel}(H)(S, R)$ *for an arbitrary higher-order polynomial functor* H*.*

(iii) *Relation lifting for polynomial functors preserves the property of ‘having the same domain’. Precisely, let* $R_1, R_2 \subseteq X \times X$ *be relations with* $\mathsf{dom}(R_1) = \mathsf{dom}(R_2)$*. If* R_1 *and* R_2 *are partially reflexive then*

$$\mathsf{dom}(\mathrm{Rel}(F)(R_1)) \quad = \quad \mathsf{dom}(\mathrm{Rel}(F)(R_2))$$

for all polynomial functors F*.*

Proof. By induction on the structure of the functor. The induction steps have been formalised in PVS. □

Lemma 4.2 *Let* $S \subseteq Y \times Y$ *and* $R \subseteq X \times X$ *be partially reflexive relations.*

$$\overline{S +_{\mathrm{R}} R} \quad = \quad \overline{S} +_{\mathrm{R}} \overline{R} \tag{1}$$

$$\overline{S \times_{\mathrm{R}} R} \quad = \quad \overline{S} \times_{\mathrm{R}} \overline{R} \tag{2}$$

$$\overline{\mathrm{RelList}(R)} \quad = \quad \mathrm{RelList}\left(\overline{R}\right) \tag{3}$$

Proof. This lemma has been proved in PVS. The inclusion from left to right is trivial in all cases. For the other direction one has to construct suitable zigzags.

For instance, for (2), let $((y,x),(y',x')) \in \overline{S} \times_{\mathrm{R}} \overline{R}$, so $y,\ldots,y'$ and $x,\ldots,x'$ are zigzags in S and R, respectively. Then $(y,x),\ldots,(y',x),\ldots,(y',x')$ is a zigzag in $S \times_{\mathrm{R}} R$ because R and S are partially reflexive. □

For the case of exponents one can derive that $\overline{S \Rightarrow_{\mathrm{R}} R} \subseteq \overline{S} \Rightarrow_{\mathrm{R}} \overline{R}$ holds if S is partially reflexive and R is symmetric. And $\overline{S} \Rightarrow_{\mathrm{R}} \overline{R} \subseteq \overline{S \Rightarrow_{\mathrm{R}} R}$ requires that R is a partial equivalence relation. However, these stronger assumptions are not given in the context in which I use the preceding lemma.

Lemma 4.3 *Let $S_i \subseteq Y \times Y$ and $R_i \subseteq X \times X$ be two indexed families of partially reflexive relations for an arbitrary index set I. Assume that all the R_i and all the S_i have pairwise equal domains. Then*

$$\overline{\bigcup_i (S_i +_{\mathrm{R}} R_i)} = \overline{(\bigcup_i S_i) +_{\mathrm{R}} (\bigcup_i R_i)} \quad (1)$$

$$\overline{\bigcup_i (S_i \times_{\mathrm{R}} R_i)} = \overline{(\bigcup_i S_i) \times_{\mathrm{R}} (\bigcup_i R_i)} \quad (2)$$

$$\overline{\bigcup_i \mathrm{RelList}(R_i)} = \overline{\mathrm{RelList}(\bigcup_i R_i)} \quad (3)$$

Proof. Equation (1) and the inclusion from left to right in (2) and (3) follow immediately from Lemma 2.1. For (2) it remains to prove that $(\bigcup_i S_i) \times_{\mathrm{R}} (\bigcup_i R_i) \subseteq \overline{\bigcup_i (S_i \times_{\mathrm{R}} R_i)}$. So assume $y\, S_i\, y'$ and $x\, R_j\, x'$ for some $i, j \in I$, then the sequence (y,x), (y',x), (y',x') is a zigzag in $\bigcup_i (S_i \times_{\mathrm{R}} R_i)$. For (3) one proves $\mathrm{RelList}(\bigcup_i R_i) \subseteq \overline{\bigcup_i \mathrm{RelList}(R_i)}$ by induction on lists. □

Proposition 4.4 *Let K be a cartesian functor and $(R_i)_{i \in I}$ be a family of partial equivalence relations that have pairwise equal domains. Then*

$$\mathrm{Rel}(K)(\overline{\bigcup_i R_i}) = \overline{\bigcup_i \mathrm{Rel}(K)(R_i)}$$

Proof. The proof proceeds by induction on the structure of K. Let me demonstrate the case $K = \mathsf{List}(K_1)$:

$$\begin{aligned}
\mathrm{Rel}(\mathsf{List}(K_1))\big(\overline{\bigcup_i R_i}\big) &= \mathrm{RelList}\big(\overline{\bigcup_i \mathrm{Rel}(K_1)(R_i)}\big) && \text{by Ind. Hyp.} \\
&= \overline{\mathrm{RelList}\big(\bigcup_i \mathrm{Rel}(K_1)(R_i)\big)} && \text{by 4.2 (3)} \\
&= \overline{\bigcup_i \mathrm{Rel}(\mathsf{List}(K_1))(R_i)} && \text{by 4.3 (3)}
\end{aligned}$$

The side conditions are discharged with the preceding lemmas. □

The important part of the preceding proposition is the subset relation from left to right: Assume we are given a pair $(t, t') \in \mathrm{Rel}(K)(\overline{\bigcup_i R_i})$ then we can safely assume that there is a zigzag $t, \ldots, t'$ in $\bigcup_i \mathrm{Rel}(K)(R_i)$.

Lemma 4.5 *Let G be an extended cartesian functor and let $(S_i) \subseteq Y \times Y$ and $(R_i) \subseteq X \times X$ be two families of partial equivalence relations indexed by an arbitrary set I. Assume that all the S_i and all the R_i have pairwise equal domains. Let $(s, r) \in \mathrm{Rel}(G)(S_j, R_j)$ for some $j \in I$. If we have additionally that for all $i \in I$ both $(r, r) \in \mathrm{Rel}(G)(S_i, R_i)$ and $(s, s) \in \mathrm{Rel}(G)(S_i, R_i)$ then we have also $(s, r) \in \mathrm{Rel}(G)(\overline{\bigcup_i S_i}, \overline{\bigcup_i R_i})$.*

Proof. The proof proceeds by induction on the structure of G. All induction steps have been formalised in PVS. Let me demonstrate the case $G(Y,X) = K(Y) \Rightarrow G_1(Y,X)$ in detail. From the assumptions it follows that s and r are functions such that there exists $j \in I$ with

$$\forall a,b \in K(Y)\,. \quad \mathrm{Rel}(K)(S_j)(a,b) \;\; \text{implies} \;\; \mathrm{Rel}(G_1)(S_j,R_j)(s\,a, r\,b) \tag{1}$$

and that for all $i \in I$

$$\forall a,b \in K(Y)\,. \quad \mathrm{Rel}(K)(S_i)(a,b) \;\; \text{implies} \;\; \mathrm{Rel}(G_1)(S_i,R_i)(s\,a, s\,b) \tag{2}$$

$$\forall a,b \in K(Y)\,. \quad \mathrm{Rel}(K)(S_i)(a,b) \;\; \text{implies} \;\; \mathrm{Rel}(G_1)(S_i,R_i)(r\,a, r\,b) \tag{3}$$

It remains to show that for all $a,b \in K(Y)$:

$$\mathrm{Rel}(K)(\overline{\textstyle\bigcup_i S_i})(a,b) \quad \text{implies} \quad \mathrm{Rel}(G_1)(\overline{\textstyle\bigcup_i S_i},\, \overline{\textstyle\bigcup_i R_i})(s\,a,\; r\,b)$$

Assume $(a,b) \in \mathrm{Rel}(K)(\overline{\bigcup_i S_i})$, by Proposition 4.4 there is a zigzag $a,\ldots,b$ in $\bigcup_i \mathrm{Rel}(K)(S_i)$. Now one can build the sequence $s\,a,\,\ldots,s\,b,\;r\,b$ and invoke the induction hypothesis for G_1 on each two adjacent elements to show that it is a zigzag in $\mathrm{Rel}(G_1)(\overline{\bigcup_i S_i},\,\overline{\bigcup_i R_i})$. This completes the proof because the latter is a partial equivalence relation by Lemma 4.1 (ii). The assumptions of the induction hypothesis require some care, but they can be discharged with the preceding lemmas and the assumptions $(1) - (3)$. $\square$

Theorem 4.6 *Let $c : X \longrightarrow G(X,X)$ be a coalgebra for an extended cartesian functor G. The partial bisimulation equivalences for c on a fixed domain form a complete lattice. In this lattice the join of a family (R_i) of bisimulations is given by $\overline{\bigcup_i R_i}$. In particular $\overline{\bigcup_i R_i}$ is a bisimulation for any family (R_i) of bisimulation equivalences.*

Proof. It is sufficient to show that for an arbitrary indexed family (R_i) of partial bisimulation equivalences on the same domain the relation $\overline{\bigcup_i R_i}$ is a bisimulation. If R_i is a bisimulation for c then $(c,c) \in \mathrm{Rel}(X \Rightarrow G)(R_i,R_i)$. Invoking Lemma 4.5 on the collection (R_i) and the pair (c,c) yields then the required $(c,c) \in \mathrm{Rel}(X \Rightarrow G)(\overline{\bigcup_i R_i},\overline{\bigcup_i R_i})$. $\square$

The preceding result applies to the functor G'_{Buf} from Example 3.2. So if one models environments of chained buffers with association lists then a greatest bisimulation equivalence does exist for all models. The greatest bisimulation equivalence can be used, for instance, to define behavioural equality in a specification language like CCSL [12].

Example 4.7 This example shows a coalgebra for an extended polynomial functor for which there is an infinitely ascending chain of bisimulation equivalences without an upper bound. So the preceding theorem cannot be generalised without making further assumptions. Consider the extended polynomial functor

$$G(Y,X) \quad \stackrel{\text{def}}{=} \quad (\mathbb{N} \Rightarrow Y) \Rightarrow \mathsf{bool}$$

For a relation $R \subseteq U \times V$ two functions $a : \mathbb{N} \longrightarrow U$ and $b : \mathbb{N} \longrightarrow V$ are R–related if $R(a\,n, b\,n)$ holds for all $n \in \mathbb{N}$. The relation lifting for G is

$$\mathrm{Rel}(G)(R, S) \quad = \quad \big\{(f, g) \mid \text{for all } R\text{–related functions } a \text{ and } b : f\,a = g\,b\big\}$$

In the following I define a G–coalgebra c with the natural numbers $\mathbb{N}$ as state space. So c takes as second argument a function $\mathbb{N} \longrightarrow \mathbb{N}$. A relation $R \subseteq \mathbb{N} \times \mathbb{N}$ is a bisimulation for c if for all $x, y \in \mathbb{N}$ with $x\,R\,y$ and all R–related functions f and g it holds that $c(x)(f) = c(y)(g)$.

I call a function $\mathbb{N} \longrightarrow \mathbb{N}$ *bounded* if there exists a natural number n such that $f\,i < n$ for all $i \in \mathbb{N}$. The main point is, that for the total relation $\mathbb{N} \times \mathbb{N}$ there exist pairs of functions $f, g : \mathbb{N} \longrightarrow \mathbb{N}$ such that f and g are $\mathbb{N} \times \mathbb{N}$–related and f is bounded while g is not bounded. The coalgebra c is now defined as

$$c(x)(f) \quad = \quad \begin{cases} \mathsf{true} & f \text{ is bounded} \\ \mathsf{false} & \text{otherwise} \end{cases}$$

The construction of c ensures that the total relation $\mathbb{N} \times \mathbb{N}$ is not a bisimulation for c. Consider now the family $(S_n)_{n \in \mathbb{N}}$ of partial equivalence relations defined by

$$i\,S_n\,j \quad \text{if and only if} \quad i = j \;\text{ or }\; (i < n \;\wedge\; j < n)$$

For any of the S_n and two functions f and g the following holds: If f and g are S_n related then f is bounded precisely when g is. Therefore all the S_n are bisimulation equivalences. Further, any finite set of relations S_n has an upper bound, which is a bisimulation for c (see also Proposition 4.4 in [14]). The least upper bound of all S_n is the total relation, so there is no greatest bisimulation equivalence. ■

5 Finitely based Coalgebras

This section improves Theorem 4.6 by excluding coalgebras like the one in Example 4.7. The main idea is not very difficult. However, it is a bit technical to set things up for all extended polynomial functors. Let me therefore first discuss the proof of Theorem 4.6 and Example 4.7. This will (hopefully) give some intuition that helps to keep oriented in the following.

Theorem 4.6 is limited to extended cartesian functors for the following reason: In the proof of Lemma 4.5 the induction step for the exponent requires Proposition 4.4, which holds only for cartesian functors. In turn Proposition 4.4 is restricted to cartesian functors because there are situations in which the following subset relation is strict:

$$\mathrm{Eq}(A) \Rightarrow_{\mathrm{R}} \overline{\textstyle\bigcup_i R_i} \quad \supseteq \quad \overline{\textstyle\bigcup_i \left(\mathrm{Eq}(A) \Rightarrow_{\mathrm{R}} R_i\right)} \tag{†}$$

The coalgebra of Example 4.7 exploits such a situation (take now $A = \mathbb{N}$ in (†)): For two functions $f, g : \mathbb{N} \longrightarrow \mathbb{N}$ where f is bounded while g is unbounded we have $(f, g) \in \mathrm{Eq}(\mathbb{N}) \Rightarrow_{\mathrm{R}} \overline{\bigcup_n S_n}$ but $(f, g) \notin \overline{\bigcup_n \left(\mathrm{Eq}(\mathbb{N}) \Rightarrow_{\mathrm{R}} S_n\right)}$. The coalgebra

c of Example 4.7 is nasty enough to deliver different observations on the inputs f and g.

The main idea for the generalisation of Theorem 4.6 is to give a condition that excludes those coalgebras that exploit the subset relation in (†) in the same way as in Example 4.7. For reasons that become clear in the following I say that a coalgebra that fulfils this condition is *finitely based.* The first step in the development is a better understanding of the subset relation in (†).

Lemma 5.1 *Let $(R_i) \subseteq X \times X$ be an I–indexed set of partially reflexive relations. Assume that I is nonempty and that all the R_i have pairwise equal domains. Consider two functions $f, g : A \longrightarrow X$. If f and g differ only for finitely many arguments (i.e., if the set $\{a \mid f\,a \neq g\,a\}$ is finite) then it holds that*

$$(f, g) \in \big(\mathrm{Eq}(A) \Rightarrow_{\mathrm{R}} \overline{\textstyle\bigcup_i R_i}\big) \qquad \textit{implies} \qquad (f, g) \in \overline{\textstyle\bigcup_i \left(\mathrm{Eq}(A) \Rightarrow_{\mathrm{R}} R_i\right)}$$

Proof. This proof has been done in PVS. It proceeds by induction on the size of the set $\{a \mid f\,a \neq g\,a\}$. In the base case (where $f = g$) one has to prove that $(f\,a)\,R_i\,(f\,a)$ for some $i \in I$ under the assumptions that $\overline{\bigcup_i R_i}(f\,a, f\,a)$. This follows from the fact that all R_i are partially reflexive relations on the same domain.

For the induction step assume that the set $\{a \mid f\,a \neq g\,a\}$ has $n+1$ elements. Choose an element a_0 from this set and let $f_{a_0 \hookleftarrow g(a_0)}$ denote the function f updated with $g(a_0)$ at the argument a_0. The functions $f_{a_0 \hookleftarrow g(a_0)}$ and g differ in n positions so by induction there is a zigzag from $f_{a_0 \hookleftarrow g(a_0)}$ to g in $\bigcup_i (\mathrm{Eq}(A) \Rightarrow_{\mathrm{R}} R_i)$. From the assumptions one gets a zigzag from $f\,a_0$ to $g\,a_0$. Both zigzags can be concatenated to form a zigzag from f to g in $\bigcup_i (\mathrm{Eq}(A) \Rightarrow_{\mathrm{R}} R_i)$. □

The preceding result shows that for Example 4.7 it is essential that the result of the application $c(x)(f)$ depends on an infinite number of mappings of f. Assume, for the sake of the discussion, that in the context of Example 4.7 we are given a computable G–coalgebra $d : \mathbb{N} \longrightarrow (\mathbb{N} \Rightarrow \mathbb{N}) \Rightarrow \mathtt{bool}$. If d is computable then in an application $d(x)(f)$ the coalgebra d can 'evaluate' its functional argument f only for finitely many arguments. Looking back to the critical subset relation (†) assume we have a pair of functions (f, g) that is contained in the left hand side. Because the hypothetical coalgebra d 'evaluates' its functional argument only on finitely many positions one can construct a function f' such that $d(x)(f) = d(x)(f')$ and that, additionally, the set $\{n \mid f'\,n \neq g\,n\}$ is finite. With the preceding lemma we can now conclude that the pair $(d(x)(f), d(x)(g))$ is also contained in the right hand side of (†). Thus, for the hypothetical coalgebra d a greatest bisimulation equivalence *does* exist.

To formalise the idea of the preceding paragraph for all coalgebras of extended polynomial functors I need to coin a few new notions. The set of arguments, which d passes into f (for a particular x and a particular f), is called an *argument descriptor.* Such a descriptor D tells how one can change the argument f without that the coalgebra d is able to detect the changes. The set of all arguments f' that the coalgebra d cannot distinguish from f is the *environment of f with respect to the descriptor D.* The coalgebra d, which has the property, that for every x and every f there exists a finite argument descriptor, is said to be *finitely based*.

In the following I formalise these notions for arbitrary extended polynomial functors. In these definitions the interesting case is always the one about the exponent. The cases for (co–)products and lists are only required to lift the properties through an arbitrary functor. At the end of the development I prove that for finitely based coalgebras of extended polynomial functors a greatest bisimulation equivalence does exists. Theorem 4.6 will reappear as a corollary: The set of descriptors for cartesian functors will be trivial and all coalgebras for extended cartesian functors are finitely based.

Definition 5.2 [Argument descriptors] Let F be a polynomial functor. The set of argument descriptors for F, denoted with desc^F, is defined by induction on the structure of F:

$$\begin{array}{rcl} \mathsf{desc}^A & = & \mathbf{1} \quad = \quad \{*\} \\ \mathsf{desc}^{\mathsf{ID}} & = & \mathbf{1} \\ \mathsf{desc}^{\mathsf{List}(F_1)} & = & \mathsf{List}(\mathsf{desc}^{F_1}) \\ \mathsf{desc}^{F_1+F_2} & = & \mathsf{desc}^{F_1} \times \mathsf{desc}^{F_2} \\ \mathsf{desc}^{F_1\times F_2} & = & \mathsf{desc}^{F_1} \times \mathsf{desc}^{F_2} \\ \mathsf{desc}^{A\Rightarrow F_1} & = & \{R \subseteq A \times \mathsf{desc}^{F_1} \mid a\,R\,b_1 \wedge a\,R\,b_2 \ \text{ implies } \ b_1 = b_2\} \end{array}$$

A descriptor $D \in \mathsf{desc}^F$ is a structure describing possible arguments that can be passed into an element of $F(X)$. If F is a cartesian functor, then ‘passing an argument’ makes no sense. Therefore desc^K is trivial for cartesian functors K. Actually, the set desc^K is isomorphic to $\mathbf{1}$ for cartesian functors that do not contain List as an ingredient functor. In case $F(X) = \mathsf{List}(F_1(X))$ a descriptor $D \in \mathsf{desc}^F$ contains a descriptor for every element in a list $l \in F(X)$. It would be nice if a descriptor D that is used in conjunction with a list l is precisely as long as l. However, it is technically easier to let the following definition accommodate for the case where D is shorter than l.

In case $F(X) = A \Rightarrow F_1(X)$ a descriptor $D \in \mathsf{desc}^F$ contains a set of arguments that can be passed into $f \in F(X)$. For each such argument a the descriptor D contains precisely one (sub–)descriptor for $f(x)$. The sets $\mathsf{desc}^{A\times B\Rightarrow F}$ and $\mathsf{desc}^{A\Rightarrow B\Rightarrow F}$ are not isomorphic as one would expect. However, they are isomorphic if one restricts $\mathsf{desc}^{A\Rightarrow B\Rightarrow F}$ to those pairs (a, R) for which $R \in \mathsf{desc}^{B\Rightarrow F}$ is nonempty.

An argument descriptor D is *hereditarily finite*, if D itself is finite and if all elements of D are hereditarily finite. All descriptors for cartesian functors are hereditarily finite.

For a fixed polynomial functor F an argument descriptor $D \in \mathsf{desc}^F$ gives rise to an environment[4] $\mathsf{env}^D(x) \subseteq F(X)$ for every element $x \in F(X)$. This environment contains all those elements $y \in F(X)$ that cannot be distinguished

[4] There is no connection to topology here.

from x when evaluation is restricted to what is contained in the argument descriptor D.

Definition 5.3 [Environment with respect to an argument descriptor] Let F be a polynomial functor and let D be an argument descriptor in desc^F. For an element $x \in F(X)$ the environment of x with respect to D is denoted with $\mathsf{env}^D(x)$ and defined by induction on the structure of F:

F	$\mathsf{env}^D(x) \;\subseteq\; F(X)$
A	$\{x\}$
X	$\{x\}$
$F_1 \times F_2$	$\mathsf{env}^{\pi_1 D}(\pi_1\, x) \;\times\; \mathsf{env}^{\pi_2 D}(\pi_2\, x)$
$F_1 + F_2$	if $x = \kappa_1\, x_1$: $\mathsf{env}^{\pi_1 D}(x_1)$ if $x = \kappa_2\, x_2$: $\mathsf{env}^{\pi_2 D}(x_2)$
$\mathsf{List}(F_1)$	if $x = \mathsf{nil}$: $\{\mathsf{nil}\}$ if $\left\{ \begin{array}{l} x = \mathsf{cons}(a_1, x') \wedge \\ D = \mathsf{cons}(A, D') \end{array} \right\}$ $\{\mathsf{cons}(a_2, y') \mid a_2 \in \mathsf{env}^A(a_1) \wedge\; y' \in \mathsf{env}^{D'}(x')\}$ if $D = \mathsf{nil}$: $\{y \mid \mathsf{length}\, x = \mathsf{length}\, y\}$
$A \Rightarrow F_1$	$\{y \mid \forall (a, D') \in D \,.\, y\, a \in \mathsf{env}^{D'}(x\, a)\}$

For each x and D we have $x \in \mathsf{env}^D(x)$. Further, for all cartesian functors K and all elements $x \in K(X)$ there exists a (hereditarily finite) descriptor $D \in \mathsf{desc}^K$ such that $\mathsf{env}^D(x) = \{x\}$.

It is now possible to formulate a particular generalisation of Proposition 4.4 for polynomial functors.

Proposition 5.4 *Let $(R_i) \subseteq X \times X$ be an I–indexed family of partially reflexive relations for a nonempty index set I. Assume that all of the R_i have pairwise equal domains. Then, for $x, y \in F(X)$ and all hereditarily finite descriptors $D \in \mathsf{desc}^F$ there exists $x' \in \mathsf{env}^D(x)$ such that*

$$(x, y) \;\in\; \mathrm{Rel}(F)\big(\overline{\textstyle\bigcup_i R_i}\big) \qquad \textit{implies} \qquad (x', y) \;\in\; \overline{\textstyle\bigcup_i \mathsf{Rel}(F)(R_i)}$$

Proof. The proof proceeds by induction on the structure of F. The induction steps for product, coproduct, and list are similar to the proof of Proposition 4.4. The induction step for the exponent relies on Lemma 5.1. All induction steps have been formalised in PVS. □

Definition 5.5 [Finitely Based] Let G be an extended polynomial functor. The property of being *finitely based* is a predicate on $G(Y, X)$. It is defined by induction on the structure of G. An element $x \in G(Y, X)$ is finitely based if and only if:

- $G(Y, X) = X$: all $x \in G(Y, X)$ are finitely based.
- $G(Y, X) = A$: all $x \in G(Y, X)$ are finitely based.
- $G(Y, X) = \mathsf{List}(G_1(Y, X))$: all elements of the list x are finitely based.
- $G(Y, X) = G_1(Y, X) \times G_2(Y, X)$: if both $\pi_1\, x$ and $\pi_2\, x$ are finitely based.
- $G(Y, X) = G_1(Y, X) + G_2(Y, X)$: if $x = \kappa_1\, x_1$ then x_1 must be finitely based; similarly for the second injection.
- $G(Y, X) = F(Y) \Rightarrow G_1(Y, X)$: for all $a \in F(Y)$ the result $x(a)$ is finitely based and further there exists an hereditarily finite $D \in \mathsf{desc}^F$ such that $\forall a' \in \mathsf{env}^D(a)\, .\, x\, a = x\, a'$.

A G–coalgebra $c : X \longrightarrow G(X, X)$ is finitely based if it is finitely based when considered as an element of $G'(X, X) = X \Rightarrow G(X, X)$. This is the case if, for each possible argument a that can be passed into c, there exists a hereditarily finite argument descriptor D such that c cannot distinguish between a and elements from the environment of a. Basically, this means that c does only inspect a 'finite part' of the argument a. There are no restrictions for arguments whose type corresponds to a cartesian functor, because for cartesian functors every argument descriptor is hereditarily finite. Therefore coalgebras for extended cartesian functors are always finitely based. In general it is undecidable if a given coalgebra is finitely based. The following result is a generalisation of Lemma 4.5.

Lemma 5.6 *Let G be an extended polynomial functor and let $(S_i) \subseteq Y \times Y$ and $(R_i) \subseteq X \times X$ be two I–indexed families of partial equivalence relations. Assume that I is nonempty and that all the S_i and all the R_i have pairwise equal domains. Let $(s, r) \in \mathrm{Rel}(G)(S_j, R_j)$ for some $j \in I$ and assume that s is finitely based. If we have additionally that for all $i \in I$ both $(r, r) \in \mathrm{Rel}(G)(S_i, R_i)$ and $(s, s) \in \mathrm{Rel}(G)(S_i, R_i)$ then we have also $(s, r) \in \mathrm{Rel}(G)(\overline{\bigcup_i S_i}, \overline{\bigcup_i R_i})$.*

Proof. The proof is almost identical to the proof of Lemma 4.5 with a little change in the induction step for the exponent. There one has to show the following implication for all $a, b \in F(Y)$:

$$\mathrm{Rel}(F)(\overline{\textstyle\bigcup_i S_i})(a, b) \quad \text{implies} \quad \mathrm{Rel}(G_1)(\overline{\textstyle\bigcup_i S_i},\, \overline{\textstyle\bigcup_i R_i})(s\, a,\, r\, b)$$

Fix a and b with $(a, b) \in \mathrm{Rel}(F)(\overline{\bigcup_i S_i})$. Because s is finitely based there exists an hereditarily finite descriptor $D \in \mathsf{desc}^F$ for a. By Proposition 5.4 there exists $a' \in \mathsf{env}^D(a)$ with $s\, a = s\, a'$ such that $(a', b) \in \overline{\bigcup_i \mathsf{Rel}(F)(S_i)}$. Now the argument continues as in the proof of Lemma 4.5. $\square$

Theorem 5.7 *Let $c : X \longrightarrow G(X, X)$ be a coalgebra for an extended polynomial functor G. If c is finitely based then the partial bisimulation equivalences for c on a fixed domain form a complete lattice. The join of a family (R_i) of bisimulations in this lattice is $\overline{\bigcup_i R_i}$.*

Proof. Apply Lemma 5.6 as in the proof of Theorem 4.6. $\square$

Example 5.8 Consider a coalgebra $c : X \longrightarrow G_{\mathsf{Buf}}(X, X)$ for the chained buffers, which models environments as functions $\mathbb{N} \longrightarrow X + \mathbf{1}$. To work out what it means for c to be finitely based one has to consider only G_{Buf}'s third ingredient that stems from the method push. Set $\mathsf{push}_c(x, e) = \pi_3(c\,x)(e)$ then c is finitely based if the following is true.

$$
\begin{aligned}
&\forall x : X \,.\, \forall e : \mathbb{N} \longrightarrow X + 1 \,.\, \exists A \subseteq \mathbb{N} \,.\, A \text{ is finite} \quad \wedge \\
&\qquad \forall e' : \mathbb{N} \longrightarrow X + 1 \,.\, e|_A = e'|_A \quad \text{implies} \quad \mathsf{push}_c(x, e) = \mathsf{push}_c(x, e') \qquad (\ddagger)
\end{aligned}
$$

This condition looks a bit complicated, but all what it says is that the result of $\mathsf{push}_c(x, e)$ depends only on a finite number of addresses in e. For all G_{Buf}–coalgebras that fulfil (‡) a greatest bisimulation equivalence does exist. The condition (‡) contains a small problem: An implementation of push that returns the argument environment (i.e., $\mathsf{push}(x, e) = e$) does not fulfil (‡). But this can be fixed by interpreting the result of push as an environment that contains only those buffers that have been changed by this method invocation (so for the resulting environment one must merge $\mathsf{push}(x, e)$ into e). ■

6 Conclusion

This paper presents two sufficient conditions for the existence of greatest bisimulations for coalgebraic signatures that contain arbitrary method types, including binary methods, in the framework of [14]. This generalises the results presented in [14,10].

The first condition in Theorem 4.6 is a purely structural one: For signatures that can be modelled with extended cartesian functors partial bisimulation equivalences on the same domain form a complete lattice.

The second one in Theorem 5.7 has weaker structural requirements: the signature must fit into larger class of extended polynomial functors. But in turn coalgebras are required to be finitely based. Being finitely based is a weak requirement: it excludes only coalgebras that are not computable functions.

All functors in this paper can contain the functor $\mathsf{List} : \mathbf{Set} \longrightarrow \mathbf{Set}$ as an ingredient functor. To put it differently, the two conditions explicitly allow the type constructor for lists in coalgebraic signatures. The obvious extension to cover all (strictly covariant) abstract data types remains for future work.

An important limitation of the results of the present paper is that they do not apply to bisimulations $R \subseteq X \times Y$ between *two* coalgebras. The main problem here is to find an appropriate generalisation of the notion of partial equivalence relations to relations $R \subseteq X \times Y$. For transitivity such a generalisation might look as follows: A relation $R \subseteq X \times Y$ is transitive with respect to a relation $Q \subseteq X \times Y$ if the following holds for all x, y, and z: if $x\, R\, y$ and $y'\, R\, z$ with $y\ Q^{\mathrm{op}}\ y'$ then also $x\, R\, z$. Note that a relation $S \subseteq X \times X$ is transitive (in the traditional sense) if and only if it is transitive with respect to the relation $\mathrm{Eq}(X)$.

Acknowledgements. I thank the anonymous referees for their comments.

References

[1] Aczel, P. and P. F. Mendler, *A final coalgebra theorem*, in: D. H. Pitt, D. E. Rydeheard, P. Dybjer, A. M. Pitts and A. Poigné, editors, *Proceedings of the Conference on Category Theory and Computer Science*, LNCS **389** (1989), pp. 357–365.

[2] Bruce, K., L. Cardelli, G. Castagna, T. H. O. Group, G. T. Leavens and B. Pierce, *On binary methods*, Theory and Practice of Object Systems **1** (1995), pp. 221–242.

[3] Hennicker, R. and A. Kurz, (Ω, Ξ)*–Logic: On the algebraic extension of coalgebraic specifications*, in: B. Jacobs and J. Rutten, editors, *Coalgebraic Methods in Computer Science '99*, ENTCS **19** (1999), pp. 195–212.

[4] Hensel, U. and B. Jacobs, *Proof principles for datatypes with iterated recursion*, in: E. Moggi and G. Rosolini, editors, *Category Theory and Computer Science*, Lecture Notes in Computer Science **1290** (1997), pp. 220–241.

[5] Hermida, C. and B. Jacobs, *Structural induction and coinduction in a fibrational setting*, Information and Computation **145** (1998), pp. 107–152.

[6] Jacobs, B., *Inheritance and cofree constructions*, in: P. Cointe, editor, *Proceedings ECOOP '96*, LNCS **1098** (1996), pp. 210–231.

[7] Jacobs, B., *Objects and classes, co–algebraically*, in: B. Freitag, C. Jones, C. Lengauer and H.-J. Schek, editors, *Object–Orientation with Parallelism and Peristence*, Kluwer Acad. Publ., 1996 pp. 83–103.

[8] Jacobs, B., "Categorical Logic and Type Theory," Studies in Logic and the Foundations of Mathematics **141**, North Holland, Elsevier, 1999.

[9] Owre, S., S. Rajan, J. Rushby, N. Shankar and M. Srivas, *PVS: Combining specification, proof checking, and model checking*, in: R. Alur and T. Henzinger, editors, *Computer Aided Verification*, Lecture Notes in Computer Science **1102** (1996), pp. 411–414.

[10] Poll, E. and J. Zwanenburg, *From algebras and coalgebras to dialgebras*, in: A. Corradini, M. Lenisa and U. Montanari, editors, *Coalgebraic Methods in Computer Science '01*, ENTCS **44** (2001).

[11] Reichel, H., *An approach to object semantics based on terminal co–algebras*, Mathematical Structure in Computer Science **5** (1995), pp. 129–152.

[12] Rothe, J., H. Tews and B. Jacobs, *The coalgebraic class specification language CCSL*, Journal of Universal Computer Science **7** (2001), pp. 175–193.

[13] Rutten, J. J. M. M., *Universal coalgebra: A theory of systems*, Theoretical Computer Science **249** (2000), pp. 3–80.

[14] Tews, H., *Coalgebras for binary methods: Properties of bisimulations and invariants*, Theoretical informatics and applications **35** (2001), pp. 83–111.