Coalgebras for Binary Methods

Hendrik Tews

Inst. Theor. Informatik
TU Dresden
D-01062 Dresden, Germany

Abstract

Coalgebras for endofunctors $\mathcal{C} \longrightarrow \mathcal{C}$ can be used to model classes of object oriented languages. However, binary methods do not fit directly into this approach. This paper proposes an extension of the coalgebraic framework, namely the use of *extended polynomial functors* $\mathcal{C}^{op} \times \mathcal{C} \longrightarrow \mathcal{C}$. This extension allows the incorporation of binary methods into coalgebraic class specifications. The paper also discusses how to define bisimulation for coalgebras of extended polynomial functors and proves some standard results.

1 Introduction

In object oriented programming an operation is called a *binary method* if it takes an additional argument of its hosting class. Consider the declaration of a class of (functional) points in a plane:

class Point
  methods
    get_x : **Self** $\longrightarrow$ real
    get_y : **Self** $\longrightarrow$ real
    move : **Self** $\times$ real $\times$ real $\longrightarrow$ **Self**
    equal : **Self** $\times$ **Self** $\longrightarrow$ bool
end Point

In contrast to usual practice in object oriented programming the implicit argument of the hosting class of a method is here explicitly denoted with **Self**. So the method move is applied to an object of class Point, takes another two arguments of type real and returns the changed state of the current object. The method equal is usually called a binary method because it takes a second argument of its hosting class.

By a standard abuse of terminology also n–ary methods, which take more than two arguments of **Self**, are referred to as binary methods [3]. This paper attempts a further generalization of the term binary method: Also higher

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

order methods, where **Self** occurs in the type of the higher order argument, are called binary methods.

Assume for an example, that for some operations we consider small finite sets of points as belonging together. Such a set could be the neighborhood of a point. Of course, the class Point should contain more methods that allow to group points to neighborhoods together and to apply operations to whole neighborhoods. To keep the example small I only consider one such method here.

$$\mathsf{move_with_neighbor} : \mathbf{Self} \times (\mathsf{Addr} \longrightarrow \mathbf{Self}) \times \mathsf{real} \times \mathsf{real} \longrightarrow (\mathsf{Addr} \longrightarrow \mathbf{Self})$$

The method move_with_neighbor is supposed to move a point together with all its assigned neighbors. To do this, it must somehow access and change the state of the other points. I assume here that they are stored under unique addresses. So the type $\mathsf{Addr} \longrightarrow \mathbf{Self}$ stands for the state of a whole plane. Via its second argument the method move_with_neighbor can access and change some of the points in the plane. As result the method delivers a new state of the plane.

The common characteristic of binary methods is, that their types contain at least two occurrences of **Self** in contravariant position.

This paper introduces the classes of *higher order polynomial functors* and *extended polynomial functors*. They are a generalization of polynomial functors from [11,21]. It defines suitable notions of coalgebra, coalgebra morphism, and bisimulation and discusses their properties. Coalgebras for higher order polynomial functors can be used to give semantics to class specifications that contain binary methods (like the example above). A first result is that the Aczel/Mendler approach to define bisimulation [1] can not be used for higher order polynomial functors. It yields a notion of bisimulation that is not closed under taking successor states (see Example 4.9 for details). The approach of Hermida and Jacobs [9] yields a bisimulation that corresponds to the intuitive notion of behavioral equivalence.

The expressiveness of higher order polynomial functors prevents useful results, which are known to hold for polynomial functors. For example, bisimulations for higher order polynomial functors are not closed under intersection (see Example 4.7). Extended polynomial functors are a proper subclass of higher order polynomial functors. They seem to be a good compromise in expressiveness: while they can model binary methods that are used in practice, their associated notion of bisimulation behaves much more nicely. Bisimulations for extended polynomial functors are closed under intersection (Proposition 5.3) and the graph of a morphism is a bisimulation (Proposition 5.6). As a last result this paper shows that final coalgebras do not exist for all extended polynomial functors.

This paper extends earlier work on coalgebraic specification [8,11], where polynomial functors are used to model class signatures. The extension is necessary because polynomial functors are not expressive enough to model

binary methods.

Other related work is that of Kurz and Hennicker. In [6] they describe algebraic extensions for coalgebraic specifications. Although it was probably not their primary motivation, it is possible to formalize certain binary methods as an algebraic extension in the setting of [6]. But there is a severe restriction: only binary methods whose codomain equals **Self** can be turned into algebraic extensions. So the method equal from the example above does not fit into the framework of Kurz and Hennicker.

A different approach to coalgebraic specification using coequations is presented in [4]. There, Cîrstea presents a sound and complete deduction calculus for a restricted set of coalgebras. Binary methods do not fit into her notion of destructor signatures. The work presented in this paper is not so much concerned about deduction calculi. The primary goal here is to make it possible to include binary methods in coalgebraic specifications.

The difficulties in treating binary methods are not restricted to the field of coalgebraic specification. In [3] Bruce et al. summarize the results with respect to binary methods of at least twenty years of research in type theory for object oriented languages. The difficulties in this field stem from the desire to assign types to classes such that inheritance gives rise to a corresponding subtype relation. For binary methods this conflicts with the contravariant nature of the function type. Without going further into details, one can summarize that, although many approaches have been proposed, a completely satisfying solution has not been found (yet).

For the field of coalgebraic specification binary methods are problematic, because endofunctors are not sufficient to describe the signatures of classes with binary methods. An obvious solution is to separate co– and contravariant occurrences of **Self** and to use bifunctors $\mathcal{C}^{op} \times \mathcal{C} \longrightarrow \mathcal{C}$ to describe signatures of classes. But then it is necessary to give new definitions for terms like coalgebra, coalgebra morphism, invariance and bisimulation. The resulting notions are more complicated and behave less nicely than those for endofunctors described in [21]. This paper is a first step to explore the more difficult area.

This research started in the setting of fibred category theory to find a class of functors that can model signatures for binary methods. Additionally the relation lifting for those functors should be fibred. It soon turned out, that for the class of extended polynomial functors the desired properties could only be proved under strong assumptions like the Axiom of Choice. I decided therefore to formulate all the definitions and results in the more familiar setting of set theory. But under the surface notions from fibred category theory play an important rôle for some proofs. Therefore I make the following compromise: I use some notions of fibred category theory in an informal way, without giving their definitions here. Instead I refer the interested reader to [12]. In all formal arguments I use the concrete incarnations of the abstract notions for the setting of this paper. So this paper can be read without any background

in fibred category theory. But readers who do have this background will be able to see the connection between [7,9] and this work.

I should note that almost all proofs in this paper have been developed and checked with PVS [18]. This applies especially to all proofs of Section 2 and all the (counter–)examples of Section 3. Since it is not practical to formalize polynomial functors in PVS, I could not directly prove the main theorems about extended polynomial functors in Section 5. But for all the results that are proved by induction on the structure of extended polynomial functors, I used PVS to formalize the inductive predicate and all the induction steps. The PVS sources are available via http://wwwtcs.inf.tu-dresden.de/~tews/binary/.

This paper is structured as follows: After introducing some less familiar notation in Section 2 I define a category of coalgebras for higher order polynomial functors in Section 3. Section 4 presents relation lifting for higher order polynomial functors and the definition of bisimulation. This section discusses also the Aczel/Mendler approach to define bisimulation and shows that this does not yield a satisfactory notion of bisimulation. Section 5 introduces the class of extended polynomial functors and proves some interesting properties. I conclude in Section 6 and show directions for further research.

I would like to thank Bart Jacobs for many discussions on the subject and my friends in Dresden for suggestions to improve the presentation of this paper. Part of this work was performed during my visit to the University of Nijmegen, the Netherlands. I want to thank the group of Frits Vaandrager at this university for their support and continuous effort to integrate myself into the Dutch social life.

2 Preliminaries

This section introduces the notation for the rest of this paper and states some general results. I will not give an introduction into the field of coalgebraic specification here. Instead I refer the reader to [14] for a general introduction of coalgebras and coinduction. Many examples of coalgebras and general results can be found in [21]. Coalgebraic specification is presented in [4,6,7] and [11].

Let me start to fix some notation. As I already said in the introduction, all results in this paper apply to the category of sets and total functions, which I denote as **Sets**. I use $\times$ to denote the Cartesian product with the projections $X \xleftarrow{\pi_1} X \times Y \xrightarrow{\pi_2} Y$ and $\langle f, g\rangle : A \longrightarrow X \times Y$ for the tuple of two functions $f : A \longrightarrow X$ and $g : A \longrightarrow Y$. I use $+$ for the coproduct (disjoint union) with injections $X \xrightarrow{\kappa_1} X + Y \xleftarrow{\kappa_2} Y$ and the cotuple $[f, g] : X + Y \longrightarrow Z$ for functions $f : X \longrightarrow Z$ and $g : Y \longrightarrow Z$. Finally I write $X \Rightarrow Y$ for the function space (exponent) between X and Y.

For a function $f : X \longrightarrow Y$ I denote its inverse image with respect to a predicate $Q \subseteq Y$ on Y with $f^*(Q) \subseteq X = \{x \in X \mid f(x) \in Q\}$. The image

of f with respect to a predicate $P \subseteq X$ is $\coprod_f(P) \subseteq Y = \{f(x) \mid x \in P\}$.[1] In some examples I use lambda calculus notation [2] and write $\lambda a : A \,.\, f(a)$ for a function with domain A that maps every $a \in A$ to $f(a)$. The identity function is $\mathrm{id}_A = \lambda a : A \,.\, a$ and the composition of two functions $f : X \longrightarrow Y$ and $g : Y \longrightarrow Z$ is $g \circ f = \lambda x : X \,.\, g(f(x))$.

I use the term *polynomial functor* as in [11] and [21]. There an endofunctor F on **Sets** is called polynomial if one of the cases hold

$$F(X) \quad = \quad \begin{cases} A \\ X \\ F_1(X) \times F_2(X) \\ F_1(X) + F_2(X) \\ A \Rightarrow F_1(X) \end{cases}$$

Here A is an arbitrary constant set and F_1 and F_2 are previously defined polynomial functors. A coalgebra for F is a function $X \longrightarrow F(X)$.

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 $\mathsf{Eq}(A) = \{(a,a) \mid a \in A\}$. Occasionally I write $=_A$ when the type of both sides of an equation is not obvious from the context. The composition of two relations $R \subseteq X \times Y$, $S \subseteq Y \times Z$ is $R \circ S \subseteq X \times Z$.

For relations there exists a special Cartesian structure. To avoid confusion I use $\times_R, +_R$ and $\Rightarrow_R$ for the relational product, coproduct and exponent. For two relations $S \subseteq U \times V$ and $R \subseteq X \times Y$ this structure is given by

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

Remark 2.1 It is possible to turn relations into the following category. Objects are (typed) relations $S \subseteq U \times V$ and $R \subseteq X \times Y$ and morphisms $S \longrightarrow R$ are pairs of functions $(f : U \longrightarrow X, g : V \longrightarrow Y)$ such that $S(u,v)$ implies $R(f(u), g(v))$. This category of relations is in fact fibred over **Sets** $\times$ **Sets**. It can be obtained by change of base from the subset fibration over **Sets**. It is Cartesian closed with the relational product, coproduct and exponent. Their action on morphisms can be defined with $(u,v) \odot_{\mathrm{R}} (f,g) = ((u \odot f), (v \odot g))$

[1] The notation f^* is borrowed from fibred category theory. Here it is the substitution functor for f in the subset fibration of **Sets**. And $\coprod_f$ is its left adjoint.

for $\odot \in \{\times, +, \Rightarrow\}$ and suitable functions u, v, f and g (see [7] for a more detailed presentation).

The fibred structure of this category of relations is not necessary for this presentation. For the rest of this paper I disregard this additional structure and use $\times_{\mathrm{R}}, +_{\mathrm{R}}$ and $\Rightarrow_{\mathrm{R}}$ purely as functions on relations.

The remainder of this section presents some technical results about the relational product, coproduct and exponent. Except from Lemma 2.5 these results are well known and can be found for instance in [7,12].

Lemma 2.2

(i) *Monotonicity. Assume four relations $S_1, S_2 \subseteq U \times V$ and $R_1, R_2 \subseteq X \times Y$. Then $S_1 \subseteq S_2$ and $R_1 \subseteq R_2$ implies*

$$\begin{array}{rcl} S_1 \times_{\mathrm{R}} R_1 & \subseteq & S_2 \times_{\mathrm{R}} R_2 \\ S_1 +_{\mathrm{R}} R_1 & \subseteq & S_2 +_{\mathrm{R}} R_2 \\ S_2 \Rightarrow_{\mathrm{R}} R_1 & \subseteq & S_1 \Rightarrow_{\mathrm{R}} R_2 \end{array}$$

Note, that the exponent is anti–monotone in its contravariant position.

(ii) *Equality. For arbitrary sets A and B*

$$\begin{array}{rcl} \mathrm{Eq}(A) \times_{\mathrm{R}} \mathrm{Eq}(B) & = & \mathrm{Eq}(A \times B) \\ \mathrm{Eq}(A) +_{\mathrm{R}} \mathrm{Eq}(B) & = & \mathrm{Eq}(A + B) \\ \mathrm{Eq}(A) \Rightarrow_{\mathrm{R}} \mathrm{Eq}(B) & = & \mathrm{Eq}(A \Rightarrow B) \end{array}$$

(iii) *Opposite Relation. For arbitrary relations $S \subseteq U \times V$, $R \subseteq X \times Y$:*

$$\begin{array}{rcl} (S \times_{\mathrm{R}} R)^{op} & = & S^{op} \times_{\mathrm{R}} R^{op} \\ (S +_{\mathrm{R}} R)^{op} & = & S^{op} +_{\mathrm{R}} R^{op} \\ (S \Rightarrow_{\mathrm{R}} R)^{op} & = & S^{op} \Rightarrow_{\mathrm{R}} R^{op} \end{array}$$

(iv) *Intersection. For a set A, relations $S_1, S_2 \subseteq U \times V$ and $R_1, R_2 \subseteq X \times Y$:*

$$\begin{array}{rcl} (S_1 \times_{\mathrm{R}} R_1) \cap (S_2 \times_{\mathrm{R}} R_2) & = & (S_1 \cap S_2) \times_{\mathrm{R}} (R_1 \cap R_2) \\ (S_1 +_{\mathrm{R}} R_1) \cap (S_2 +_{\mathrm{R}} R_2) & = & (S_1 \cap S_2) +_{\mathrm{R}} (R_1 \cap R_2) \\ (\mathrm{Eq}(A) \Rightarrow_{\mathrm{R}} R_1) \cap (\mathrm{Eq}(A) \Rightarrow_{\mathrm{R}} R_2) & = & \mathrm{Eq}(A) \Rightarrow_{\mathrm{R}} (R_1 \cap R_2) \\ (S_1 \Rightarrow_{\mathrm{R}} R_1) \cap (S_2 \Rightarrow_{\mathrm{R}} R_2) & \subseteq & (S_1 \cap S_2) \Rightarrow_{\mathrm{R}} (R_1 \cap R_2) \quad (*) \end{array}$$

(v) *Union. For a set A, relations $S_1, S_2 \subseteq U \times V$ and $R_1, R_2 \subseteq X \times Y$:*

$$\begin{array}{rcl} (S_1 \times_{\mathrm{R}} R_1) \cup (S_2 \times_{\mathrm{R}} R_2) & \subseteq & (S_1 \cup S_2) \times_{\mathrm{R}} (R_1 \cup R_2) \quad (*) \\ (S_1 +_{\mathrm{R}} R_1) \cup (S_2 +_{\mathrm{R}} R_2) & = & (S_1 \cup S_2) +_{\mathrm{R}} (R_1 \cup R_2) \\ (\mathrm{Eq}(A) \Rightarrow_{\mathrm{R}} R_1) \cup (\mathrm{Eq}(A) \Rightarrow_{\mathrm{R}} R_2) & \subseteq & \mathrm{Eq}(A) \Rightarrow_{\mathrm{R}} (R_1 \cup R_2) \quad (*) \end{array}$$

(vi) *Composition. Assume four relations* $S_1 \subseteq U \times V$, $S_2 \subseteq V \times W$, $R_1 \subseteq X \times Y$, $R_2 \subseteq Y \times Z$ *and a set* A*:*

$$\begin{array}{rcll}
(S_1 \times_{\mathrm{R}} R_1) \circ (S_2 \times_{\mathrm{R}} R_2) & = & (S_1 \circ S_2) \times_{\mathrm{R}} (R_1 \circ R_2) & \\
(S_1 +_{\mathrm{R}} R_1) \circ (S_2 +_{\mathrm{R}} R_2) & = & (S_1 \circ S_2) +_{\mathrm{R}} (R_1 \circ R_2) & \\
(\mathrm{Eq}(A) \Rightarrow_{\mathrm{R}} R_1) \circ (\mathrm{Eq}(A) \Rightarrow_{\mathrm{R}} R_2) & = & \mathrm{Eq}(A) \Rightarrow_{\mathrm{R}} (R_1 \circ R_2) & \\
(S_1 \Rightarrow_{\mathrm{R}} R_1) \circ (S_2 \Rightarrow_{\mathrm{R}} R_2) & \subseteq & (S_1 \circ S_2) \Rightarrow_{\mathrm{R}} (R_1 \circ R_2) & (*)
\end{array}$$

Remark 2.3 The lines tagged with $(*)$ in the previous Lemma state a subset relation instead of an equality. For these cases the equality does not hold in general. The PVS formalization of this paper contains examples for which these subset relations are strict.

Item (v) does not contain a statement about the general exponent. The two expressions in question $(R_1 \cup R_2) \Rightarrow_{\mathrm{R}} (S_1 \cup S_2)$ and $(R_1 \Rightarrow_{\mathrm{R}} S_1) \cup (R_1 \Rightarrow_{\mathrm{R}} S_2)$ are not comparable at all in general. The items (iv) (intersection) and (v) (union) can be generalized to arbitrary collections except for the second equation of (iv), which only holds for nonempty collections.

Proof. All results have been proved within PVS. They are obtained by tedious but easy computations. The third equation of 2.2 (vi) (composition) requires the Axiom of Choice. □

An important notion in fibred category theory is that of a (co–)fibred functor. As a technical condition fibred and cofibred functors play an important rôle for some results in Section 5. But instead of developing the definition of a fibred functor here, I use its defining property in all formal arguments: An endofunctor H on the category of relations from remark 2.1 is fibred over an endofunctor K on **Sets** if for all $R \subseteq X \times Y$ it is the case that $H(R) \subseteq K(X) \times K(Y)$. Additionally, for all functions $f : X' \longrightarrow X$, $g : Y' \longrightarrow Y$ the equation $H((f \times g)^* R) = (K(f) \times K(g))^* H(R)$ must hold. If H and K are contravariant, then the required equation is $H((f \times g)^* R) = \coprod_{K(f) \times K(g)} H(R)$.

The next result is well known and can be found for instance in [7]. I specialize it to the setting of this paper.

Lemma 2.4 *The relational product, coproduct and exponent are fibred. For all relations* $S \subseteq U \times V$ *and* $R \subseteq X \times Y$ *and functions* $u : U' \longrightarrow U$, $v : V' \longrightarrow V$, $f : X' \longrightarrow X$, $g : Y' \longrightarrow Y$ *it holds that*

(i) $\left((u \times v)^* S\right) \times_{\mathrm{R}} \left((f \times g)^* R\right) = ((u \times f) \times (v \times g))^* (S \times_{\mathrm{R}} R)$

(ii) $\left((u \times v)^* S\right) +_{\mathrm{R}} \left((f \times g)^* R\right) = ((u + f) \times (v + g))^* (S +_{\mathrm{R}} R)$

(iii) *And for* $k : U \longrightarrow U'$, $h : V \longrightarrow V'$*:*
$\left(\coprod_{(k \times h)} S\right) \Rightarrow_{\mathrm{R}} \left((f \times g)^* R\right) = ((k \Rightarrow f) \times (h \Rightarrow g))^* (S \Rightarrow_{\mathrm{R}} R)$

Proof. This Lemma has been proved in PVS. The results follow from general properties of the category **Sets** and they are immediate, after unpacking the

abstract notation. For the exponent we have for instance

$$\begin{aligned}
\big(\coprod_{(k\times h)} S\big) \Rightarrow_{\mathrm{R}} \big((f\times g)^*\, R\big) \quad &\subseteq \quad (U' \Rightarrow X') \times (V' \Rightarrow Y') \\
= \quad & \big\{(a,b) \mid \forall p' : U', q' : V' \,. \\
& \qquad \big(\exists p : U, q : V \,.\, p' = k(p) \;\text{ and }\; q' = h(q) \;\text{ and }\; S(p,q)\big) \\
& \qquad\quad \text{implies }\; R(\, f(a(p')), g(b(q'))\,)\big\} \\
= \quad & \big\{(a,b) \mid \forall p : U, q : V \,.\, S(p,q) \;\text{ implies }\; R\big(f(a(k(p))), g(b(h(q)))\big)\big\} \\
= \quad & ((k \Rightarrow f) \times (h \Rightarrow g))^*\, (S \Rightarrow_{\mathrm{R}} R)
\end{aligned}$$

□

The notion of a cofibred functor is similar to that of a fibred functor: For cofibredness f^* and $\coprod_f$ exchange their rôle. A functor H on relations is cofibred over K, if $H(R) \subseteq K(X) \times K(Y)$ and $H(\coprod_{f\times g} R) = \coprod_{K(f)\times K(g)} H(R)$ for a relation $R \subseteq X \times Y$ and functions $f : X \longrightarrow X'$, $g : Y \longrightarrow Y'$.

It is a new observation that it is impossible to get a result like Lemma 2.4 for cofibredness. Although product and coproduct are cofibred, for the exponent cofibredness fails. The PVS formalization of this paper contains an example for relations $S \subseteq U \times V$, $R \subseteq X \times Y$ and functions $u : U' \longrightarrow U$, $v : V' \longrightarrow V$, $f : X \longrightarrow X'$ and $g : Y \longrightarrow Y'$ such that $\big((u\times v)^*\, S\big) \Rightarrow_{\mathrm{R}} \big(\coprod_{f\times g} R\big) \neq \coprod_{(u\Rightarrow f)\times(v\Rightarrow g)} (S \Rightarrow_{\mathrm{R}} R)$. However it is possible to recover cofibredness for the exponent in the special case of constant arguments.

Lemma 2.5 *Assume relations $S \subseteq U \times V$ and $R \subseteq X \times Y$ and functions $u : U \longrightarrow U', v : V \longrightarrow V', f : X \longrightarrow X', g : Y \longrightarrow Y'$. Then*

(i) $\big(\coprod_{u\times v} S\big) \times_{\mathrm{R}} \big(\coprod_{f\times g} R\big) \quad = \quad \coprod_{(u\times f)\times(v\times g)} (S \times_{\mathrm{R}} R)$

(ii) $\big(\coprod_{u\times v} S\big) +_{\mathrm{R}} \big(\coprod_{f\times g} R\big) \quad = \quad \coprod_{(u+f)\times(v+g)} (S +_{\mathrm{R}} R)$

(iii) *For an arbitrary set A*
$\mathrm{Eq}(A) \Rightarrow_{\mathrm{R}} \coprod_{f\times g} R \quad = \quad \coprod_{(\mathrm{id}_A\Rightarrow f)\times(\mathrm{id}_A\Rightarrow g)} \big(\mathrm{Eq}(A) \Rightarrow_{\mathrm{R}} R\big)$

Proof. This Lemma has been proved in PVS. Points (1) and (2) are again easy calculations. For (3) the left hand side is

$$\begin{aligned}
\mathrm{Eq}(A) \Rightarrow_{\mathrm{R}} \coprod_{f\times g} R \quad &\subseteq \quad (A \Rightarrow X') \times (A \Rightarrow Y') \\
= \quad & \big\{(h',k') \mid \forall a : A \,.\, \exists x : X, y : Y \,.\, R(x,y) \text{ and} \\
& \qquad h'(a) = f(x) \text{ and } k'(a) = g(y)\big\} \qquad (*)
\end{aligned}$$

And the right hand side

$$\begin{aligned}
&\coprod_{(\mathrm{id}_A\Rightarrow f)\times(\mathrm{id}_A\Rightarrow g)} \big(\mathrm{Eq}(A) \Rightarrow_{\mathrm{R}} R\big) \\
&\quad = \quad \big\{(h',k') \mid \exists h : A \longrightarrow X, k : A \longrightarrow Y \,. \\
&\qquad\quad \forall a : A \,.\, R(h(a), k(a)) \text{ and } h' = f \circ h \text{ and } k' = g \circ k\big\} \qquad (\dagger)
\end{aligned}$$

There is an obvious inclusion from (†) to (∗). For the other direction it is necessary to construct suitable functions h and k using the Axiom of Choice.□

3 Higher order polynomial functors

This section introduces the class of higher order polynomial functors and a generalized notion of coalgebra. Higer order polynomial functors extend polynomial functors with arbitrary exponents to allow for class signatures with binary methods.

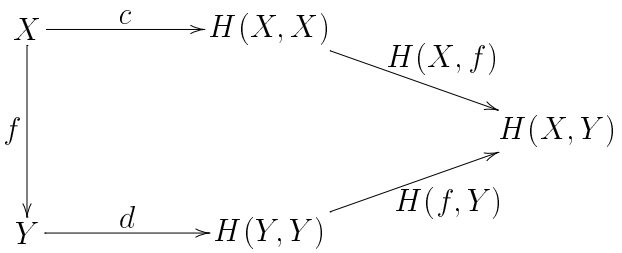
Definition 3.1 [Higher order polynomial functors]
A functor $H : \mathbf{Sets}^{op} \times \mathbf{Sets} \longrightarrow \mathbf{Sets}$ is called a *higher order polynomial functor* if it is defined as one of the cases

$$H(Y,X) \quad = \quad \begin{cases} A \\ X \\ H_1(Y,X) \times H_2(Y,X) \\ H_1(Y,X) + H_2(Y,X) \\ H_1(X,Y) \Rightarrow H_2(Y,X) \end{cases}$$

where A is an arbitrary constant set and H_1 and H_2 are previously defined higher order polynomial functors. The morphism part is defined in the obvious way.

Higher order polynomial functors work on two arguments. This is necessary to separate occurrences of the arguments with positive and negative variance. Both arguments are therefore swapped on the left hand side of $\Rightarrow$.

Definition 3.2 [Category of Coalgebras]
Let H be a higher order polynomial functor. An 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 called H–coalgebra morphism $f : c \longrightarrow d$, if the following diagram commutes:



This defines a category of H–coalgebras for each higher order polynomial functor H.

Remark 3.3 Every polynomial functor is also a higher order polynomial functor. If a higher order functor H is equivalent to a polynomial functor

F (i.e. if $H(Y,X) = F(X)$ for all X and Y) then the above pentagon collapses to the familiar square

$$\begin{array}{ccc} X & \xrightarrow{\quad c \quad} & F(X) \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle F(f)} \\ Y & \xrightarrow{\quad d \quad} & F(Y) \end{array}$$

Example 3.4 Coalgebras of higher order polynomial functors can be used to model classes of object oriented languages. Binary methods do not pose problems any longer. For the class of points from the introduction one gets the functor

$$\mathsf{PointIface}(Y,X) = \begin{cases} \mathbb{R} \times \mathbb{R} \quad \times \\ (\mathbb{R} \times \mathbb{R} \longrightarrow X) \quad \times \\ (Y \longrightarrow \mathsf{bool}) \quad \times \\ \Big((A \longrightarrow Y) \times \mathbb{R} \times \mathbb{R} \longrightarrow (A \longrightarrow X) \Big) \end{cases}$$

Here, bool is the set of booleans, $\mathbb{R}$ is used for the set of the real numbers and A is the set of addresses.

A specific class that realizes the interface of points corresponds to a coalgebra $c : X \longrightarrow \mathsf{PointIface}(X,X)$. We get the single methods as projections:

$$\begin{aligned} \mathsf{get_x}_c &= \pi_1 \circ c \\ \mathsf{get_y}_c &= \pi_2 \circ c \\ \mathsf{move}_c &= \pi_3 \circ c \\ \mathsf{equal}_c &= \pi_4 \circ c \\ \mathsf{move_with_neighbor}_c &= \pi_5 \circ c \end{aligned}$$

If $d : Y \longrightarrow \mathsf{PointIface}(Y,Y)$ is another $\mathsf{PointIface}$–coalgebra then a function $f : X \longrightarrow Y$ is a point morphism $c \longrightarrow d$ precisely if for all $x, x' \in X$, $r_1, r_2 \in \mathbb{R}$ and $e : A \longrightarrow X$ it holds that [2]

$$\begin{aligned} \mathsf{get_x}_c(x) &= \mathsf{get_x}_d(f(x)) \\ \mathsf{get_y}_c(x) &= \mathsf{get_y}_d(f(x)) \\ f\big(\mathsf{move}_c(x, r_1, r_2)\big) &= \mathsf{move}_d(f(x), r_1, r_2) \\ \mathsf{equal}_c(x, x') &= \mathsf{equal}_d(f(x), f(x')) \\ \lambda a \,.\, f\big(\mathsf{move_with_neighbor}_c(x, e, r_1, r_2)(a)\big) &= \\ \mathsf{move_with_neighbor}_d\big(f(x),\ (\lambda a \,.\, f(e(a))), r_1, r_2\big) & \end{aligned}$$

[2] In the following equations I use currying and write $move(x, r_1, r_2)$ instead of $move(x)(r_1, r_2)$ to get a better presentation.

4 Bisimulation for higher order polynomial functors

Bisimulations are used in various process calculi (for example in [17]) and in the field of coalgebras to capture behavioral equivalence. For coalgebras it is necessary to have a definition for bisimulation that is parametric in the functor. There are two traditions: Following Aczel and Mendler in [1], a bisimulation is the state space of a coalgebra that makes a certain diagram (of coalgebra morphisms) commute. Rutten, Kurz and Hennicker follow this approach in [21] and [6]. In the following I will call this the Aczel/Mendler approach and when necessary I will use the term *Aczel/Mendler bisimulation*.

The second traditions stems from Hermida and Jacobs. In [9] they define a special operation —called relation lifting— for polynomial functors. Relation lifting is then used to define bisimulations. This approach is used for instance in [7,8,11]. When necessary I use the term *Hermida/Jacobs bisimulation* to avoid confusion.

Both approaches have their advantages. Without discussing this in full detail, I only want to note that the Aczel/Mendler approach applies to all endofunctors and not just to polynomials. Relation lifting has a more complex definition on first sight, but is easier to use in practice (because it is not necessary to construct a coalgebra acting on the bisimulation).

For higher order polynomial functors the Aczel/Mendler approach does not capture the intuitive notion of behavioral equivalence. It is possible to construct an Aczel/Mendler bisimulation R that relates two states such that their successor states are not related by R and may give different observations (see Example 4.9 below).

This paper follows therefore Hermida and Jacobs and defines bisimulation via relation lifting.

Definition 4.1 [Relation lifting] Let $S \subseteq V \times Y$ and $R \subseteq U \times X$ be two relations and let H be 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:

- If $H(Y,X) = X$, then

$$\mathrm{Rel}(H)(S,R) \quad = \quad R.$$

- If $H(Y,X) = A$ (for A a constant set), then

$$\mathrm{Rel}(H)(S,R) \quad = \quad \mathrm{Eq}(A) \quad = \quad \{(a,a) \mid a \in A\}.$$

- If $H(Y,X) = H_1(Y,X) \times H_2(Y,X)$, then

$$\begin{array}{rcl} \mathrm{Rel}(H)(S,R) & = & \mathrm{Rel}(H_1)(S,R) \times_{\mathrm{R}} \mathrm{Rel}(H_2)(S,R) \\ & = & \{((x_1,x_2),(y_1,y_2)) \mid \mathrm{Rel}(H_1)(S,R)(x_1,y_1) \\ & & \qquad \text{and } \mathrm{Rel}(H_2)(S,R)(x_2,y_2)\}. \end{array}$$

- If $H(Y,X) = H_1(Y,X) + H_2(Y,X)$, then

$$\begin{array}{rcl} \mathrm{Rel}(H)(S,R) & = & \mathrm{Rel}(H_1)(S,R) +_{\mathrm{R}} \mathrm{Rel}(H_2)(S,R) \\ & = & \{(\kappa_1\, x_1, \kappa_1\, y_1) \mid \mathrm{Rel}(H_1)(S,R)(x_1,y_1)\} \\ & & \quad \cup \{(\kappa_2\, x_2, \kappa_2\, y_2) \mid \mathrm{Rel}(H_2)(S,R)(x_2,y_2)\}. \end{array}$$

- If $H(Y,X) = H_1(X,Y) \Rightarrow H_2(Y,X)$, then

$$\begin{array}{rcl} \mathrm{Rel}(H)(S,R) & = & \mathrm{Rel}(H_1)(R,S) \Rightarrow_{\mathrm{R}} \mathrm{Rel}(H_2)(S,R) \\ & = & \big\{(f,g) \mid f : H_1(U,V) \longrightarrow H_2(V,U)\,, g : H_1(X,Y) \longrightarrow H_2(Y,X) \\ & & \quad \text{such that } \forall a \in H_1(U,V), b \in H_1(X,Y)\,.\, \mathrm{Rel}(H_1)(R,S)(a,b) \\ & & \quad\quad \text{implies } \mathrm{Rel}(H_2)(S,R)(f(a),g(b)) \big\}. \end{array}$$

Relation lifting for higher order polynomial functors works on two argument relations. In $\mathrm{Rel}(H)(S,R)$ the relation S is used for the contravariant occurrences of Y in $H(Y,X)$. The relation R is for the covariant argument of H. As you can see, relation lifting simply substitutes $\times_{\mathrm{R}}, +_{\mathrm{R}}, \Rightarrow_{\mathrm{R}}$ for $\times, +, \Rightarrow$ in the structure of H and uses equality on constants. The effect is that two elements $s \in H(V,U)$, $t \in H(Y,X)$ are in $\mathrm{Rel}(H)(S,R)$ only if they have the same structure. This means that, if $H = H_1 + H_2$ then s and t come either both from the first component or both from the second. Further $\mathrm{Rel}(H)(S,R)$ requires S, R and equality to hold pointwise for the contravariant argument, covariant argument and constants, respectively.

This definition of relation lifting is a conservative extension of similar definitions in [9,11]. If H is equivalent to a polynomial functor F then $\mathrm{Rel}(H)(S,R) = R^F$, where the latter denotes relation lifting in [11]. Note, that for a polynomial functor F the argument relation S in $\mathrm{Rel}(F)(S,R)$ is never used, so that $\mathrm{Rel}(F)(S,R) = \mathrm{Rel}(F)(S',R)$ for all relations S'. I will therefore sometimes use $\mathrm{Rel}(F)(R)$ instead of $\mathrm{Rel}(F)(S,R)$ for polynomial functors.

Relation lifting has a close relative: predicate lifting. Predicate lifting can be used to define invariants (or subsystems as they are called in [21]). Greatest and least invariance can give semantics to modal operators like always or sometimes for coalgebras. For details see [11,13,20]. From definition 4.1 it is obvious how to extend predicate lifting for higher order polynomial functors. But this is not relevant for this paper and will be described elsewhere.

I need the following result below.

Lemma 4.2 *Let H be a higher order polynomial functor.*

(i) *Relation lifting commutes with equality:*

$$\mathit{Rel}(H)(\mathrm{Eq}(X), \mathrm{Eq}(Y)) \quad = \quad \mathrm{Eq}(H(X,Y))$$

(ii) *For two relations $S \subseteq U \times V$ and $R \subseteq X \times Y$ relation lifting commutes*

with $(-)^{op}$ *in the following sense:*

$$Rel(H)(S^{op}, R^{op}) \quad = \quad (\, Rel(H)(S,R)\,)^{op}$$

It is impossible to obtain similar results for the commutation of relation lifting with $\cap, \cup$ or relational composition. This fails because it is not possible to obtain equations instead of the subset relations in Lemma 2.2. Relation lifting for higher order polynomial functors is also not fibred, because this would require relation lifting to be cofibred, which fails for the exponent.

Proof. Both results are obtained by induction on the structure of H, using Lemmas 2.2 (ii) and 2.2 (iii) for the induction steps. □

Definition 4.3 [Bisimulation] Let H be a higher order polynomial functor and $c : X \longrightarrow H(X,X)$ and $d : Y \longrightarrow H(Y,Y)$ be two H–coalgebras. A relation $R \subseteq X \times Y$ is a H–bisimulation for c and d if for all $x \in X, y \in Y$

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

Example 4.4 Continuing the example of points, I assume again two point coalgebras $c : X \longrightarrow \mathsf{PointIface}(X,X)$ and $d : Y \longrightarrow \mathsf{PointIface}(Y,Y)$. Let us now look under which condition a relation $R \subseteq X \times Y$ forms a point bisimulation. We first need to lift a relation to the type of planes, which are the second argument of the move_with_neighbor method: Let us call two planes $e_1 : A \longrightarrow X$ and $e_2 : A \longrightarrow Y$ R–related if for all $a \in A$ we have $R(e_1(a), e_2(a))$.[3] Now the relation $R \subseteq X \times Y$ is a point bisimulation if for all $x \in X$ and $y \in Y$ with $R(x,y)$ all of the following items hold:

- $\mathsf{get_x}_c(x) \;=_{\mathbb{R}}\; \mathsf{get_x}_d(y)$
- $\mathsf{get_y}_c(x) \;=_{\mathbb{R}}\; \mathsf{get_y}_d(y)$
- $\forall r_1, r_2 \in \mathbb{R}\,.\; R\big(\mathsf{move}_c(x, r_1, r_2),\; \mathsf{move}_d(y, r_1, r_2)\big)$
- $\forall x' \in X, y' \in Y\,.\, R(x', y')$ implies $\mathsf{equal}_c(x, x') \;=_{\mathsf{bool}}\; \mathsf{equal}_d(y, y')$
- Forall R–related planes $e_1 : A \longrightarrow X$ and $e_2 : A \longrightarrow Y$ and all $r_1, r_2 \in \mathbb{R}$ we have that also the results $\mathsf{move_with_neighbor}_c(x, e_1, r_1, r_2)$ and $\mathsf{move_with_neighbor}_d(y, e_2, r_1, r_2)$ are R–related.

It is now the question if bisimulations for higher order polynomial functors enjoy similar properties like bisimulations for weak pullback preserving endofunctors do. For the more general case of higher order polynomial functors, the possibility to specify arbitrary functional arguments and the contravariant nature of the function type causes problems. So there are only a few positive results.

Lemma 4.5 *Let H be a higher order polynomial functor and $c : X \longrightarrow H(X,X)$ and $d : Y \longrightarrow H(Y,Y)$ be two H–coalgebras.*

[3] The notion of R–related planes coincides deliberately with the relation lifting $\mathrm{Rel}(A \Rightarrow X)(R,R) = \{(e_1, e_2) \mid \forall a : A\,.\, R(e_1(a), e_2(a))\}$

(i) *The equality relation* $\mathrm{Eq}(X)$ *is a bisimulation for* c.

(ii) *If* $R \subseteq X \times Y$ *is a bisimulation for* c *and* d, *then* R^{op} *is a bisimulation for* d *and* c.

Proof. Apply Lemma 4.2. □

Fact 4.6 *For higher order polynomial functors*

(i) *the intersection of two bisimulations is not always a bisimulation,*

(ii) *the union of two bisimulations is not always a bisimulation,*

(iii) *the composition of two bisimulations is not always a bisimulation,*

(iv) *the graph of a morphism is not always a bisimulation,*

(v) *and finally the kernel of a morphism is not always a bisimulation.*

Here I include only an example for two bisimulations whose intersection is not a bisimulation again. Examples for the other facts are contained in the PVS formalization of this paper. However, it is worth remarking that the examples for Items (iii), (iv), and (v) involve the same functor T of the following example. For the union of bisimulations, it is possible to use the simpler functor $H(Y, X) = Y \Rightarrow A$.

Example 4.7 Consider the functor

$$T(Y, X) \quad \stackrel{\text{def}}{=} \quad (X \Rightarrow Y) \Rightarrow X$$

For two functions $f : U \longrightarrow X$ and $g : Y \longrightarrow V$ its morphism part is

$$\begin{aligned} T(g, f) \quad &: \quad T(V, U) \longrightarrow T(Y, X) \\ &= \quad (f \Rightarrow g) \Rightarrow f \\ h : (U \longrightarrow V) \longrightarrow U \longmapsto \lambda k : X \longrightarrow Y \,.\, &(f \circ h)(g \circ k \circ f) \end{aligned}$$

To describe the relation lifting for T it is useful to define the auxiliary notion of (R, S)–related functions: For two relations $R \subseteq X \times Y$, $S \subseteq U \times V$ the functions $a : X \longrightarrow U$ and $b : Y \longrightarrow V$ are (R, S)–related if, for all $x \in X$ and $y \in Y$, it holds that $R(x, y)$ implies $S(a(x), b(y))$.

Assuming again two relations $S \subseteq U \times V$ and $R \subseteq X \times Y$ the relation lifting for T is then

$$\begin{aligned} \mathrm{Rel}(T)(S, R) &\subseteq T(U, X) \times T(V, Y) \\ &= \big\{ (f, g) \mid f : (X \longrightarrow U) \longrightarrow X \,, g : (Y \longrightarrow V) \longrightarrow Y \quad \text{such that} \\ &\qquad \text{for all } (R, S)\text{–related functions } a \text{ and } b : R\big(f(a), g(b)\big) \ \big\} \end{aligned}$$

Take now two sets $A \stackrel{\text{def}}{=} \{a_1, a_2, a_3, a_4\}$, $B \stackrel{\text{def}}{=} \{b_1, b_2, b_3, b_4\}$ and the relations

$R \stackrel{\text{def}}{=} \{(a_1, b_1), (a_2, b_2)\}$ and $S \stackrel{\text{def}}{=} \{(a_1, b_1), (a_3, b_3)\}$. Define the following functions:

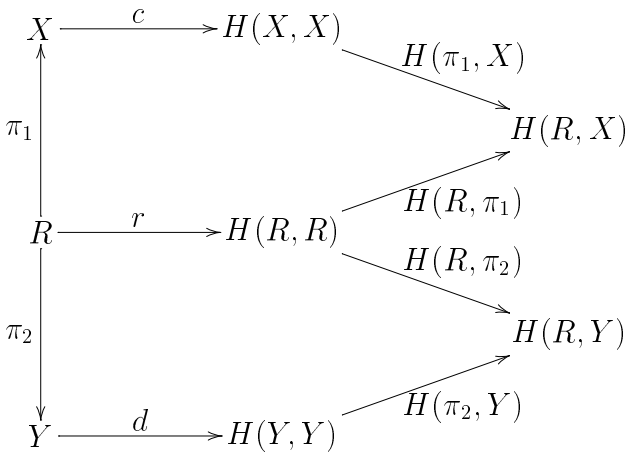
$$
\begin{aligned}
f &: A \longrightarrow A \\
&= \lambda a : A \,.\, \mathsf{if}\ a = a_1\ \mathsf{then}\ a_1\ \mathsf{else}\ a_4\ \mathsf{endif} \\
g &: B \longrightarrow B \\
&= \lambda b : B \,.\, \mathsf{if}\ b = b_1\ \mathsf{then}\ b_1\ \mathsf{else}\ b_4\ \mathsf{endif} \\
c &: A \longrightarrow (A \longrightarrow A) \longrightarrow A \\
&= \lambda a : A \,.\, \lambda h : A \longrightarrow A \,.\, \mathsf{if}\ h = f\ \mathsf{then}\ a_4\ \mathsf{else}\ a_1\ \mathsf{endif} \\
d &: B \longrightarrow (B \longrightarrow B) \longrightarrow B \\
&= \lambda b : B \,.\, \lambda k : B \longrightarrow B \,.\, \mathsf{if}\ k = g\ \mathsf{then}\ b_4\ \mathsf{else}\ b_1\ \mathsf{endif}
\end{aligned}
$$

The functions f and g are neither (R, R)–related nor (S, S) related. But they are $(R \cap S, R \cap S)$–related and $(c(a_1)(f), d(b_1)(g)) \notin R \cap S$. That is why R and S are bisimulations for c and d, but $R \cap S$ is not. So bisimulations are not closed under intersection.

One could question now the relevance of this example because for any T–coalgebra $Z \longrightarrow T(Z, Z)$ the universal relation $Z \times Z$ is a bisimulation. The functor T does also have a final coalgebra. But the argument above applies also to more complicated functors like $T'(Y, X) = (X \Rightarrow Y) \Rightarrow (X \times A)$.

In the remainder of this subsection I will define a second notion of bisimulation, following the approach of Aczel and Mendler. It will turn out, that this gives a different notion of bisimulation (compared to Definition 4.3). To avoid confusion I use the term *Hermida/Jacobs bisimulation* for bisimulations in the sense of Definition 4.3 for the rest of this section.

Definition 4.8 [Aczel/Mendler bisimulation] Let $c : X \longrightarrow H(X, X)$ and $d : Y \longrightarrow H(Y, Y)$ be coalgebras for a higher order polynomial functor H. A relation $R \subseteq X \times Y$ is called an *Aczel/Mendler bisimulation* if there exists a coalgebra $r : R \longrightarrow H(R, R)$ such that $\pi_1 : R \longrightarrow X$ and $\pi_2 : R \longrightarrow Y$ are H–coalgebra morphisms, *i.e.* if the following diagram commutes



The notions of Aczel/Mendler bisimulation and Hermida/Jacobs bisimulation are not related in general. It is not the case, that the Aczel/Mendler bisimulation subsumes the Hermida/Jacobs bisimulation and also the converse is not true. The following example shows a relation R that is an Aczel/Mendler bisimulation but fails to fulfill Definition 4.3. An example for the converse situation uses the same functor and is contained in the PVS formalization of this paper.

Example 4.9 Consider again the functor T from Example 4.7. Take now for a concrete state space the set $M \stackrel{\text{def}}{=} \{m_1, m_2\}$ and let R be the relation that relates only m_1 with itself: $R = \{(m_1, m_1)\}$. Define the following T–coalgebras

$$\begin{aligned} c &: M \longrightarrow (M \rightarrow M) \rightarrow M \\ &= \lambda x : M \,.\, \lambda a : M \rightarrow M \,.\, \mathsf{if}\ a = \mathrm{id}_M\ \mathsf{then}\ m_2\ \mathsf{else}\ m_1\ \mathsf{endif} \\ r &: R \longrightarrow (R \rightarrow R) \rightarrow R \\ &= \lambda r : R \,.\, \lambda r' : R \rightarrow R \,.\, (m_1, m_1) \end{aligned}$$

The question is now if R is a bisimulation for c (to instantiate Definition 4.8 one has to take $X = Y = M$ and $c = d$). Intuitively we should expect that R is not a T–bisimulation for c, because for the state m_1 we get $c(m_1)(\mathrm{id}_M) = m_2$. So if $(m_1, m_1) \in R$ then R should also contain the pair (m_2, m_2), because a bisimulation should be closed under taking successor states. And indeed, since id_M is (R, R)–related with itself, we find that $(c(m_1), c(m_1)) \notin \mathrm{Rel}(T)(R, R)$ and R is not a Hermida/Jacobs bisimulation.

Checking for the Aczel/Mendler bisimulation we find that

$$\begin{aligned} T(\pi_1, R) \circ c &: M \longrightarrow (M \rightarrow R) \rightarrow M \\ &= \lambda x : M \,.\, \lambda a : M \rightarrow R \,.\, m_1 \end{aligned}$$

and r is indeed a coalgebra fulfilling the condition of Definition 4.8.

This shows that the Aczel/Mendler bisimulation is not "sound": it is possible to relate states with a bisimulation, which are intuitively not behaviorally equivalent.

5 Extended polynomial functors

With the negative results of the last section it is very difficult to imagine applications of coalgebras for higher order polynomial functors. A careful look at the example 4.7 shows that the source of all the problems is a general exponent occurring with negative variance.

If we take a look at object oriented languages then we find that neither Java [5] nor Eiffel [16] allow parameters for methods (or features as they are called in Eiffel) to be of functional type. So for these languages higher order polynomial functors are far too general. Arguments of functional type are also

impossible in C++ [22], but there one can use pointers to functions to pass functions as arguments to other functions. However, there are languages that encourage the user to use functional arguments. One example is Ocaml [15].

The class Point in the introduction contains an application for arguments of functional type. But even there the type of the argument is Addr⟶**Self** and **Self** occurs only with strictly positive variance in the type of the functional argument. It is very hard to find a convincing example, where one method takes an argument of type **Self**⟶**Self**. Therefore I suggest the following restriction.

Definition 5.1 [Extended polynomial functors]
A functor $G : \mathbf{Sets}^{op} \times \mathbf{Sets} \longrightarrow \mathbf{Sets}$ is called *extended polynomial* if it is built according to the grammar

$$G(Y,X) \quad = \quad \begin{cases} A \\ X \\ G_1(Y,X) \times G_2(Y,X) \\ G_1(Y,X) + G_2(Y,X) \\ G_1(A',Y) \Rightarrow G_2(Y,X) \end{cases}$$

where A, A' are arbitrary constant sets and G_1 and G_2 are previously defined extended polynomial functors.

The only (but crucial) difference compared to higher order polynomial functors is the clause for the exponent. This accounts for the following fact: If G is an extended polynomial functor then for each set A there is a polynomial functor F such that $F(X) = G(A,X)$.

The functor T from example 4.7 is not an extended polynomial functor. But the functor PointIface (Example 3.4) fits into Definition 5.1. For object oriented specification there is the following rule of thumb: A class signature gives rise to an extended polynomial functor if all its binary methods have first order arguments (like the method equal) or for every functional argument, it is the case that **Self** does only occur in strictly covariant position in the type of the argument.

Because every extended polynomial functor is also a higher order polynomial functor, extended polynomial functors inherit the definitions of coalgebra (Definition 3.2), relation lifting (Definition 4.1) and bisimulation (Definition 4.3) from higher order polynomial functors. But because of the restricted exponent these notions behave much more nicely.

The remainder of this section lists the results, which I have been able to prove so far.

Lemma 5.2 *Let G be an extended polynomial functor.*

(i) *Assume additionally four relations $R_1, R_2 \subseteq X \times Y$ and $S_1, S_2 \subseteq U \times V$.*

Then

$$Rel(G)(S_1, R_1) \;\cap\; Rel(G)(S_2, R_2) \quad\subseteq\quad Rel(G)(S_1 \cap S_2, R_1 \cap R_2)$$

(ii) *Assume now relations* $R_1 \subseteq X \times Y$, $R_2 \subseteq Y \times Z$, $S_1 \subseteq U \times V$ *and* $S_2 \subseteq V \times W$

$$Rel(G)(S_1, R_1) \;\circ\; Rel(G)(S_2, R_2) \quad\subseteq\quad Rel(G)(S_1 \circ S_2, R_1 \circ R_2)$$

Proof. For (i) I first prove for polynomial functors F

$$\mathrm{Rel}(F)(R_1) \;\cap\; \mathrm{Rel}(F)(R_2) \quad=\quad \mathrm{Rel}(F)(R_1 \cap R_2) \qquad (*)$$

by induction on their structure. The two base cases are obvious and for the induction steps I use the first three equations from Lemma 2.2 (iv). Let me do the induction step for $F(X) = A \Rightarrow F_1(X)$ in detail. Assuming as induction hypothesis $\mathrm{Rel}(F_1)(R_1) \cap \mathrm{Rel}(F_1)(R_2) = \mathrm{Rel}(F_1)(R_1 \cap R_2)$ I compute

$$\begin{aligned}
&\mathrm{Rel}(A \Rightarrow F_1)(R_1) \;\cap\; \mathrm{Rel}(A \Rightarrow F_1)(R_2) \\
&\quad= \quad \big(\mathrm{Eq}(A) \Rightarrow_{\mathrm{R}} \mathrm{Rel}(F_1)(R_1)\big) \;\cap\; \big(\mathrm{Eq}(A) \Rightarrow_{\mathrm{R}} \mathrm{Rel}(F_1)(R_2)\big) \\
&\quad= \quad \mathrm{Eq}(A) \Rightarrow_{\mathrm{R}} \big(\mathrm{Rel}(F_1)(R_1) \;\cap\; \mathrm{Rel}(F_1)(R_2)\big) && \text{by 2.2 (iv)} \\
&\quad= \quad \mathrm{Eq}(A) \Rightarrow_{\mathrm{R}} \mathrm{Rel}(F_1)(R_1 \cap R_2) && \text{by Ind. Hyp.} \\
&\quad= \quad \mathrm{Rel}(A \Rightarrow F_1)(R_1 \cap R_2)
\end{aligned}$$

For the result 5.2 (i) I do induction on the structure of G. To apply the induction hypothesis I need now additionally Lemma 2.2 (i). I do again the most difficult induction step for $G(Y, X) = G_1(A, Y) \Rightarrow G_2(Y, X)$ in detail. Assume that $G_1(A, -)$ is equivalent to the polynomial functor F. Then

$$\begin{aligned}
&\mathrm{Rel}(F \Rightarrow G_2)(S_1, R_1) \;\cap\; \mathrm{Rel}(F \Rightarrow G_2)(S_2, R_2) \\
&\quad= \quad \big(\mathrm{Rel}(F)(S_1) \Rightarrow_{\mathrm{R}} \mathrm{Rel}(G_2)(S_1, R_1)\big) \;\cap \\
&\qquad\qquad \big(\mathrm{Rel}(F)(S_2) \Rightarrow_{\mathrm{R}} \mathrm{Rel}(G_2)(S_2, R_2)\big) \\
&\quad\subseteq \quad \big(\mathrm{Rel}(F)(S_1) \cap \mathrm{Rel}(F)(S_2)\big) \;\Rightarrow_{\mathrm{R}} && \text{by 2.2 (iv)} \\
&\qquad\qquad \big(\mathrm{Rel}(G_2)(S_1, R_1) \cap \mathrm{Rel}(G_2)(S_2, R_2)\big) \\
&\quad= \quad \mathrm{Rel}(F)(S_1 \cap S_2) \;\Rightarrow_{\mathrm{R}} && \text{by } (*) \\
&\qquad\qquad \big(\mathrm{Rel}(G_2)(S_1, R_1) \cap \mathrm{Rel}(G_2)(S_2, R_2)\big) \\
&\quad\subseteq \quad \mathrm{Rel}(F)(S_1 \cap S_2) \;\Rightarrow_{\mathrm{R}} && \text{by Ind. Hyp. and 2.2 (i)} \\
&\qquad\qquad \mathrm{Rel}(G_2)(S_1 \cap S_2, R_1 \cap R_2) \\
&\quad= \quad \mathrm{Rel}(F \Rightarrow G_2)(S_1 \cap S_2, R_1 \cap R_2)
\end{aligned}$$

For (ii) I use the same proof method: Using Lemma 2.2 (vi) I establish first an equality for polynomial functors. I can then prove the main result with induction on the structure of extended polynomial functors. □

Proposition 5.3 *Bisimulations for extended polynomial functors are closed under intersection and composition.*

Note, that bisimulations for extended polynomial functors are not closed under union in general. An example is contained in the PVS formalization of this paper.

Proof. Both proofs are very similar. I only do composition here. I have to show that, for an arbitrary extended polynomial functor G, three G–coalgebras $c : X \longrightarrow G(X,X)$, $d : Y \longrightarrow G(Y,Y)$, $e : Z \longrightarrow G(Z,Z)$, a G–bisimulation $R \subseteq X \times Y$ for c and d and a G–bisimulation $S \subseteq Y \times Z$ for d and e, also their composition $R \circ S$ is a G–bisimulation for c and e. With the definition of bisimulation it remains to show, that for all $x \in X$, $y \in Y$, $z \in Z$ such that $R(x,y)$, $S(y,z)$, $\mathrm{Rel}(G)(R,R)(c(x),d(y))$ and $\mathrm{Rel}(G)(S,S)(d(y),e(z))$ it is also the case, that $\mathrm{Rel}(G)(R \circ S, R \circ S)(c(x),e(z))$. But this follows directly from the previous Lemma. □

The next two lemmas lead to Proposition 5.6 about bisimulations and morphisms.

Lemma 5.4 *The relation lifting for polynomial functors is cofibred: for a polynomial functor F, a relation $R \subseteq X \times Y$ and two functions $f : X \longrightarrow X'$, $g : Y \longrightarrow Y'$ it holds that*

$$\mathit{Rel}(F)(\coprod_{f \times g} R) \quad = \quad \coprod_{F(f) \times F(g)} \mathit{Rel}(F)(R)$$

The relation lifting for polynomial functors is also fibred, but this is not needed explicitly here, it will follow from Lemma 5.5 below.

Proof. The proof goes by induction on the structure of F. The base cases are easy computations. For the induction steps Lemma 2.5 is used. I demonstrate the case $F(X) = A \Rightarrow F_1(X)$:

$$\begin{array}{lll}
\mathrm{Rel}(A \Rightarrow F_1)(\coprod_{f \times g} R) & & \\
\quad = \quad \mathrm{Eq}(A) \Rightarrow_{\mathrm{R}} \mathrm{Rel}(F_1)(\coprod_{f \times g} R) & & \\
\quad = \quad \mathrm{Eq}(A) \Rightarrow_{\mathrm{R}} \coprod_{F_1(f) \times F_1(g)} \mathrm{Rel}(F_1)(R) & & \text{by Ind. Hyp.} \\
\quad = \quad \coprod_{F(f) \times F(g)} \big(\mathrm{Eq}(A) \Rightarrow_{\mathrm{R}} \mathrm{Rel}(F_1)(R)\big) & & \text{by 2.5 (iii)} \\
\quad = \quad \coprod_{F(f) \times F(g)} \mathrm{Rel}(A \Rightarrow F_1)(R) & &
\end{array}$$

□

Lemma 5.5 *The relation lifting for extended polynomial functors is fibred: for an extended polynomial functor G, two relations $S \subseteq U \times V$, $R \subseteq X \times Y$ and four functions $u : U \longrightarrow U'$, $v : V \longrightarrow V'$, $f : X' \longrightarrow X$, $g : Y' \longrightarrow Y$ it holds that*

$$\mathit{Rel}(G)(\coprod_{u \times v} S,\ (f \times g)^* R) \quad = \quad \big(G(u,f) \times G(v,g)\big)^* \mathit{Rel}(G)(S,R)$$

Proof. The proof goes again by induction on the structure of G. Lemma 2.4 is applied in the induction steps. I demonstrate the case $G(Y,X) = G_1(A,Y) \Rightarrow G_2(Y,X)$. Assume that the polynomial functor F equals $G_1(A,-)$.

$$\begin{array}{lll}
\mathrm{Rel}(F \Rightarrow G_2)(\coprod_{u\times v} S,\ (f\times g)^*\, R) & & \\
\quad = & \big(\mathrm{Rel}(F)(\coprod_{u\times v} S)\big) \Rightarrow_{\mathrm{R}} \big(\mathrm{Rel}(G_2)(\coprod_{u\times v} S, (f\times g)^*\, R)\big) & \\
\quad = & \big(\coprod_{F(u)\times F(v)} \mathrm{Rel}(F)(S)\big) \Rightarrow_{\mathrm{R}} & \text{by 5.4 and Ind. Hyp} \\
 & \qquad \big((G_2(u,f)\times G_2(v,g))^*\, \mathrm{Rel}(G_2)(S,R)\big) & \\
\quad = & (G(u,f)\times G(v,g))^*\, \big(\mathrm{Rel}(F)(S) \Rightarrow_{\mathrm{R}} \mathrm{Rel}(G_2)(S,R)\big) & \text{by 2.4 (iii)} \\
\quad = & (G(u,f)\times G(v,g))^*\, \mathrm{Rel}(F\Rightarrow G_2)(S,R) &
\end{array}$$

□

Proposition 5.6 *Let G be an extended polynomial functor and assume two G–coalgebras $c : X \longrightarrow G(X,X)$ and $dY \longrightarrow G(Y,Y)$. A function $f : X \longrightarrow Y$ is a morphism $c \longrightarrow d$ if and only if the graph of f given by $\coprod_{\mathrm{id}_X\times f} \mathrm{Eq}(X)$ is a bisimulation for c and d.*

Proof. First notice, that in **Sets** there is an equivalence

$$\coprod_{\mathrm{id}_X\times f} \mathrm{Eq}(X) \quad = \quad (f\times \mathrm{id}_Y)^*\, \mathrm{Eq}(Y) \qquad (*)$$

Using this intermediate result I compute

$$\begin{array}{lll}
\mathrm{Rel}(G)(\mathsf{graph}(f), \mathsf{graph}(f)) & & \\
\quad = & \mathrm{Rel}(G)(\coprod_{\mathrm{id}_X\times f} \mathrm{Eq}(X), (f\times \mathrm{id}_Y)^*\, \mathrm{Eq}(Y)) & \text{by } (*) \\
\quad = & G(\mathrm{id}_X\times f, f\times \mathrm{id}_Y)^*\, \mathrm{Rel}(G)(\mathrm{Eq}(X), \mathrm{Eq}(Y)) & \text{by 5.5} \\
\quad = & G(\mathrm{id}_X\times f, f\times \mathrm{id}_Y)^*\, \mathrm{Eq}(G(X,Y)) & \text{by 4.2 (i)}
\end{array}$$

□

The kernel of a function $f : X \longrightarrow Y$ is a binary relation on X and contains those pairs (x_1, x_2) for which $f(x_1) = f(x_2)$. The kernel can be characterized as $\mathsf{ker}(f) = \mathsf{graph}(f) \circ \mathsf{graph}(f)^{op}$. The next result is an immediate consequence of the Proposition 5.6 and 5.3.

Proposition 5.7 *The kernel of a morphism of G–coalgebras is a bisimulation for every extended polynomial functor G.*

As next problem I come back to the relation of the two notions of bisimulation. Example 4.9 showed only that both notions are different for the functor T. And T is not an extended polynomial functor.

Proposition 5.8 *For extended polynomial functors the notions of Aczel/Mendler bisimulation and Hermida/Jacobs bisimulation coincide.*

Proof. Assume that G is an extended polynomial functor, $c : X \longrightarrow G(X,X)$ and $d : Y \longrightarrow G(Y,Y)$ are two G–coalgebras, and $R \subseteq X \times Y$ a relation. Consider the following properties

$$\begin{array}{l}\forall x \in G(X,X),\ y \in G(Y,Y),\ r \in G(R,R)\,. \\ \quad G(R,\pi_1)(r) = G(\pi_1,X)(x) \ \text{ and } \ G(R,\pi_2)(r) = G(\pi_2,Y)(y) \\ \qquad\qquad \text{implies } \ \mathrm{Rel}(G)(R,R)(x,y)\end{array} \qquad (*)$$

$$\begin{array}{l}\forall x \in G(X,X),\ y \in G(Y,Y), \\ \quad \mathrm{Rel}(G)(R,R)(x,y) \ \text{ implies } \ \exists r \in G(R,R)\,. \\ \qquad G(R,\pi_1)(r) = G(\pi_1,X)(x) \ \text{ and } \ G(R,\pi_2)(r) = G(\pi_2,Y)(y)\end{array} \qquad (\dagger)$$

If R is an Aczel/Mendler bisimulation then $(*)$ implies that R is also a Hermida/Jacobs bisimulation. If, for the other direction, R is a Hermida/Jacobs bisimulation then $(\dagger)$ implies that for each $(a,b) \in R$ there exists a suitable element in $G(R,R)$. It is therefore possible to construct the needed function $R \longrightarrow G(R,R)$ for the Aczel/Mendler bisimulation with the Axiom of Choice.

It remains to show that $(*)$ and $(\dagger)$ hold for all extended polynomial functors G and all relations R. This can be done in one induction on the structure of G. The base cases and the induction steps for $\times_{\mathrm{R}}$ and $+_{\mathrm{R}}$ are easy computations. For the exponent, the Axiom of Choice is needed again. For this proof the predicates $(*)$ and $(\dagger)$ and the induction steps have been formalized in PVS. □

A last important question that I want to discuss is the existence of final coalgebras for extended polynomial functors. An object z in a category $\mathcal{C}$ is a final object if for any object x in $\mathcal{C}$ there exists exactly one morphism $x \longrightarrow z$. This unique morphism is usually denoted with $!_x$. Final coalgebras (in a suitable category of coalgebras) are minimal (any bisimulation for the final coalgebra is contained in the equality relation) and they realize all possible behaviors (because any other coalgebra can be embedded into the final one via the unique morphism !). Final coalgebras can give semantics to behavioral datatypes or class specifications [19].

For all bounded functors (including all polynomial functors) a final coalgebra does exist [21]. So it is natural to ask if those extended polynomial functors that are not equivalent to polynomial functors have final coalgebras. I already said above, that the functor T from Example 4.7 has a final coalgebra. In general any extended polynomial functor G for which $G(\mathbf{1},\mathbf{1})$ is isomorphic to $\mathbf{1}$ has a final coalgebra.[4] It is the only function $\mathbf{1} \longrightarrow G(\mathbf{1},\mathbf{1})$. All other extended polynomial functors have a final coalgebra only if they are naturally isomorphic to a polynomial functor. Here I prove only that the functor $G(Y,X) = Y \Rightarrow \mathsf{bool}$ does not have a final coalgebra. But the proof generalizes to any extended polynomial functor.

[4] I use $\mathbf{1} = \{\bot\}$ here to denote the final object of **Sets**.

Proposition 5.9 *The functor $G(Y, X) = Y \Rightarrow \mathsf{bool}$ does not have a final coalgebra.*

Proof. Let me denote the two elements of bool with t and f and let $\mathsf{not} : \mathsf{bool} \longrightarrow \mathsf{bool}$ be the function that maps t to f and vice versa.

Assume towards a contradiction that $z : Z \longrightarrow Z \Rightarrow \mathsf{bool}$ is a final coalgebra. Construct a new coalgebra $z' : (Z + 1) \longrightarrow (Z + 1) \Rightarrow \mathsf{bool}$ by

$$z'(x)(y) = \begin{cases} z(x')(y') & \text{if } x = \kappa_1\, x' \text{ and } y = \kappa_1\, y' \\ \mathsf{not}(z(x')(x')) & \text{if } x = \kappa_1\, x' \text{ and } y = \kappa_2\, y' \\ \mathsf{t} & \text{otherwise.} \end{cases}$$

By construction $\kappa_1 : Z \longrightarrow Z + 1$ is a coalgebra morphism $z \longrightarrow z'$. Because z is the final coalgebra, there exists $!_{z'} : Z + 1 \longrightarrow Z$ and further $!_{z'} \circ \kappa_1 = \mathsf{id}_Z$. Let $\perp$ be the only element of $\mathbf{1}$ and let $\perp_! = !_{z'}(\kappa_2\, \perp)$. Then

$$\mathsf{not}(z(\perp_!)(\perp_!)) \;=\; z'(\kappa_1 \perp_!)(\kappa_2\, \perp) \;=\; z(!_{z'}(\kappa_1\, \perp_!))(\perp_!) \;=\; z(\perp_!)(\perp_!)$$

which is clearly impossible. □

This last result has serious consequences. For coalgebras of polynomial endofunctors there exists a *coinduction principle*. It states that the bisimilarity relation for the final coalgebra is contained in the equality relation. For extended polynomial functors this principle is vacuous. First, the final coalgebra does not exist for interesting extended polynomial functors. Second, it is unclear what bisimilarity should be. It is easy to construct examples that have several maximal bisimulations, but no greatest one.

A notion of coalgebra without a corresponding notion of coinduction loses much of its original attraction. But I would like to argue here, that an important application area of coalgebras is object oriented specification. Without an appropriate treatment of binary methods, coalgebraic specification will remain an exotic area in theoretical computer science. With all the problems that were cause by binary methods in the past, it is unrealistic to hope that one can get binary methods for free.

Apart from the coinduction principle, the existence of a final model can serve two purposes. First, it tells the person who developed the specification that the specification is itself consistent. Second, she can examine the states of the final model and if she does not find any unwanted behavior, she can be sure that the specification captures the right class of models. Therefore I would like to propose that for any coalgebraic specification the developer should convince herself that the final model exists. Once she did this, she can also use an appropriate coinduction principle. An interesting question is now if one can give sufficient conditions for the existence of the final model of a coalgebraic specification. Note, that this question has not been answered yet

for coalgebraic specification with polynomial functors. Jacobs gives a general construction of the final model in [10]. But he does not discuss the question, under which conditions the constructed state space is nonempty.

6 Conclusion

This paper extends the notion of coalgebras, such that coalgebraic specifications can also contain binary methods. With respect to binary methods, the approach presented here is more expressive than the use of algebraic extensions [6]. Further it has the advantage that binary methods can be specified within the coalgebraic framework.

Extended polynomial functors seem to be a good compromise. On the one side they are expressive enough for the signatures of binary methods that occur in object oriented programming. On the other side the associated notions of bisimulation and coalgebra morphism have some properties that are familiar from polynomial functors.

This research is far from complete. More results about invariants and the relation of invariants and bisimulation are available, but did not fit into this paper. They will appear elsewhere. The most interesting question is probably, if coalgebras for extended polynomial functors without a notion of coinduction are still useful for object oriented specification and verification. On first sight notions like coalgebraic refinement [11] do not depend on the existence of final coalgebras. So it seems, that some existing techniques from coalgebraic specification can also be used for the more general notion of coalgebras of this paper. But this has to be checked carefully. A second interesting question is, under which conditions coalgebraic specifications have final models.

To make it possible to experiment with binary methods, the LOOP–tool [8] for coalgebraic specification is currently extended to work with extended polynomial functors.

References

[1] P. Aczel 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*, volume 389 of *LNCS*, pages 357–365, Berlin, September 1989. Springer.

[2] H. P. Barendregt. Lambda calculi with types. In S. Abramsky, Dov M. Gabbay, and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2. Oxford Science Publications, 1992.

[3] K. Bruce, L. Cardelli, G. Castagna, The Hopkins Object Group, G. T. Leavens, and B. Pierce. On binary methods. *Theory and Practice of Object Systems*, 1(3):221–242, 1995.

[4] C. Cîrstea. A coalgebraic equational approach to specifying observational structures. In B. Jacobs and J. Rutten, editors, *Coalgebraic Methods in Computer Science*, number 19 in ENTCS. Elsevier, Amsterdam, 1999.

[5] J. Gosling, B. Joy, and G. Steele. *The Java Language Specification*. Addison-Wesley, 1996.

[6] R. Hennicker 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*, volume 19 of *ENTCS*, pages 195–212. Elsevier, 1999.

[7] U. Hensel. *Definition and Proof Principles for Data and Processes*. PhD thesis, Univ. of Dresden, Germany, 1999.

[8] U. Hensel, M. Huisman, B. Jacobs, and H. Tews. Reasoning about classes in object–oriented languages: Logical models and tools. In Ch. Hankin, editor, *European Symposium on Programming*, number 1381 in Lecture Notes in Computer Science, pages 105–121. Springer, Berlin, 1998.

[9] C. Hermida and B. Jacobs. Structural induction and coinduction in a fibrational setting. *Inf. & Comp.*, pages 107–152, 1998.

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

[11] B. Jacobs. Invariants, bisimulations and the correctness of coalgebraic refinements. In M. Johnson, editor, *Algebraic Methodology and Software Technology*, number 1349 in Lecture Notes in Computer Science, pages 276–291. Springer, Berlin, 1997.

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

[13] B. Jacobs. The temporal logic of coalgebras via Galois algebras. Techn. Rep. CSI-R9906, Comput. Sci. Inst., Univ. of Nijmegen, 1999.

[14] B. Jacobs and J. Rutten. A tutorial on (co)algebras and (co)induction. *EATCS Bulletin*, 62:222–259, 1997.

[15] X. Leroy. *The Objective Caml system, release 2.04*, November 1999. available at URL http://caml.inria.fr/ocaml/htmlman/.

[16] B. Meyer. *Eiffel: The Language*. Prentice Hall, 1992.

[17] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[18] S. Owre, S. Rajan, J.M. Rushby, N. Shankar, and M. Srivas. PVS: Combining specification, proof checking, and model checking. In R. Alur and T.A. Henzinger, editors, *Computer Aided Verification*, number 1102 in Lecture Notes in Computer Science, pages 411–414. Springer, Berlin, 1996.

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

[20] J. Rothe. Modal logic for coalgebraic class specification. Diploma thesis, Dresden University of Technology, 2000.

[21] J. Rutten. Universal coalgebra: a theory of systems. CWI Report CS-R9652, 1996.

[22] B. Stroustrup. *The C++ Programming Language*. Addison-Wesley, 2nd rev. edition, 1994.