Theoretical Informatics and Applications Will be set by the publisher
Informatique Théorique et Applications

COALGEBRAS FOR BINARY METHODS: PROPERTIES OF BISIMULATIONS AND INVARIANTS *

HENDRIK TEWS [1]

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 and invariants for coalgebras of extended polynomial functors and proves many standard results.

1991 Mathematics Subject Classification. 03E20, 03G30, 68Q55, 68Q65.

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

Keywords and phrases: binary method, coalgebra, bisimulation, invariant, object-orientation

* *A previous version of this paper appeared as [27].*

[1] Institut für Theoretische Informatik, TU Dresden, D-01062 Dresden, Germany;
e-mail: tews@tcs.inf.tu-dresden.de

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-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.

$$\mathsf{move\_with\_neighbor} : \mathbf{Self} \times \big(\mathsf{Addr} \longrightarrow \mathbf{Self}\big) \times \mathsf{real} \times \mathsf{real} \longrightarrow \big(\mathsf{Addr} \longrightarrow \mathbf{Self}\big)$$

The idea is that the method move_with_neighbor moves 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 the other points 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** (including the implicit argument of the hosting class) in contravariant position.

If we ignore the binary methods then the interface of class Point can be modeled by an endofunctor

$$T(X) \quad = \quad \mathbb{R} \times \mathbb{R} \times (\mathbb{R} \times \mathbb{R} \Rightarrow X)$$

(where $\mathbb{R}$ stands for the real numbers and $\Rightarrow$ for the function space). A particular point class (i.e., an implementation for the methods) is a coalgebra $X \longrightarrow T(X)$ and elements in X are (states of) objects.

Binary methods are problematic in this approach. If one naively includes $(X \Rightarrow \mathsf{bool})$ for the binary method equal into the preceding functor T then the result cannot be turned into an endofunctor on **Set** anymore. 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. This paper introduces the classes of *higher-order polynomial functors* and *extended polynomial functors*. They are generalizations of polynomial functors from [12, 25]. It defines suitable notions of coalgebra, coalgebra morphism, invariant, 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] cannot be used for higher-order polynomial functors. It yields a notion of bisimulation that is not closed under taking successor states (see Example 3.9 for details). The approach of Hermida and Jacobs [10] 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 and invariants for higher-order polynomial functors are not closed under intersection (see Example 3.7). 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 notions of bisimulation and invariant behave much more nicely. For instance, bisimulations and invariants for extended polynomial functors are closed under intersection (Proposition 4.3) and the graph of a morphism is a bisimulation (Proposition 4.7). As a last result this paper shows that final coalgebras do not exist for nontrivial extended polynomial functors.

This paper extends earlier work on coalgebraic specification [9, 12], where polynomial functors are used to model class signatures. The extension is necessary because polynomial functors are not expressive enough to model binary methods. The results of this paper are the basis for the extension of the coalgebraic class specification language CCSL [24] to allow binary methods in class specifications.

Other related work is that of Hennicker and Kurz. In [7] they describe algebraic extensions for coalgebraic specifications. Binary methods whose codomain equals **Self** can be formalized as an algebraic extension. However, the method equal from the example above does not fit into the framework of Hennicker and Kurz.

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 here is not so much concerned about deduction calculi. The primary goal here is to make it possible to include binary methods in coalgebraic specifications.

Hidden algebra [5, 23] is a branch of (multi-sorted) algebraic specification in which some sorts of an algebraic signature are considered as hidden sorts on which no direct observation is possible. Hidden sorts are intended to capture the state space of automata and of classes. A severe restriction in hidden algebra is that a hidden signature contains only operations $S_1 \times \cdots \times S_n \longrightarrow S_0$, where all the S_i are sorts. So in hidden algebra one has neither structured argument types nor structured result types. The above method move_with_neighbor does not fit into a hidden signature. Using coalgebras one can model partial operations easily with coalgebras of the form $\mathsf{Self} \longrightarrow \mathsf{Self} + \mathbf{1}$. In hidden algebra one has to use subsorting.

Another difference between hidden algebra and coalgebraic specification is the approach to define behavioral equivalence. In coalgebraic specification one uses bisimulations, a notion with which one can compare the behavior of different models. Hidden algebra uses the approach of Reichel [21] of visible contexts. This allows only to compare states of one model. Bisimulations for coalgebras of polynomial functors form a complete lattice [25]. However, bisimulations for extended polynomial functors are not closed under union (Observation 3.6, but see also Proposition 4.4). In contrast, in hidden algebra one has always a greatest hidden congruence even in the presence of binary methods [23].

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).

The present research started in the setting of fibred category theory to find a class of functors that can model signatures for binary methods. Additionally the predicate and 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. Therefore I decided 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 role 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 [13]. 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 [8,10] and this work.

Almost all proofs in this paper have been developed and checked with the theorem prover PVS [19]. This applies especially to all proofs of Section 1 and all the (counter–) examples of Observation 3.6. It is not practical to formalize polynomial functors in PVS. For the main propositions of Section 4 I formalized only those proofs that are done by induction on the structure of extended polynomial functors. 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 1 I define a category of coalgebras for higher-order polynomial functors in Section 2. Section 3 presents predicate and relation lifting for higher-order polynomial functors and the definitions of bisimulation and invariant. This section also discusses the Aczel/Mendler approach to define bisimulation and invariant and shows that this approach does not yield satisfactory notions. Section 4 introduces the class of extended polynomial functors and proves many properties about bisimulations and invariants for extended polynomial functors. I conclude in Section 5 and show directions for further research. A previous version of this paper appeared as [27]; Proposition 4.4 and all the material about predicate lifting and invariants is new in this version.

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. I thank several anonymous referees of the present paper and the earlier version [27] for their detailed comments.

1. PRELIMINARIES

This section introduces the notation for the present paper and states some general results. I will not give an introduction into the field of coalgebraic specification here. For an introduction to coalgebras and coinduction see [14, 25], a coalgebraic specification language that is based on coalgebras is presented in [24]. For different approaches to coalgebraic specification see [4, 7].

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 **Set**. 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 $\mathrm{id}_A = \lambda a : A \,.\, a$, the diagonal is $\delta = \langle \mathrm{id}, \mathrm{id}\rangle = \lambda a : A \,.\, (a, a)$, 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 [12] and [25]. There, an endofunctor F on **Set** is called polynomial if one of the cases holds

$$F(X) \;=\; A \;\mid\; X \;\mid\; F_1(X) \times F_2(X) \;\mid\; F_1(X) + F_2(X) \;\mid\; A \Rightarrow F_1(X)$$

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 $X \longrightarrow F(X)$. Some authors allow additionally the (finite) powerset construction in polynomial functors to model nondeterminism. In this paper I do not consider the powerset construction, this remains future work.

A predicate on a set X is written as $P \subseteq X = \{x \mid P(x)\}$. The truth predicate on X is $\top_X = X \subseteq X$. 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 $\mathrm{Eq}(A) = \{(a, a) \mid a \in A\}$. 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 $R \circ S = \{(x, z) \mid \exists y \,.\, R(x, y) \wedge S(y, z)\} \subseteq X \times Z$.

For predicates and relations there exists a special Cartesian structure. To avoid confusion I use $\times_{\mathrm{P}}, +_{\mathrm{P}}$, and $\Rightarrow_{\mathrm{P}}$ for the Cartesian structure on predicates and $\times_{\mathrm{R}}, +_{\mathrm{R}}$ and $\Rightarrow_{\mathrm{R}}$ for the relational product, coproduct and exponent. Assume two predicates $P \subseteq X$ and $Q \subseteq Y$ then the product, the coproduct, and the exponent for predicates is given by

$$\begin{array}{rcl}
P \times_{\mathrm{P}} Q \;\subseteq\; X \times Y & = & \big\{(x, y) \mid P(x) \text{ and } Q(y)\big\} \\
P +_{\mathrm{P}} Q \;\subseteq\; X + Y & = & \{\kappa_1\, x \mid P(x)\} \;\cup\; \big\{\kappa_2\, y \mid Q(y)\big\} \\
P \Rightarrow_{\mathrm{P}} Q \;\subseteq\; X \Rightarrow Y & = & \big\{f \mid \forall x \in X \,.\, P(x) \;\text{ implies }\; Q(f(x))\big\}
\end{array}$$

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

For two relations $S \subseteq U \times V$ and $R \subseteq X \times Y$ the special Cartesian structure is given by

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

Remark 1.1. It is possible to turn predicates into to following category **Pred**. Objects of **Pred** are (typed) predicates $P \subseteq X$ and $Q \subseteq Y$ and a morphism $P \longrightarrow Q$ is a function $f : X \longrightarrow Y$ such that $P(x)$ implies $Q(f(x))$. This category of predicates is in fact fibred over **Set**, it is the subobject fibration $\begin{smallmatrix}\textbf{Pred}\\ \downarrow \\ \textbf{Set}\end{smallmatrix}$. The constructions described above make **Pred** a Cartesian closed category. The action on morphisms for the Cartesian closed structure is trivial: $f \odot_{\mathrm{P}} g \;=\; (f \odot g)$ for $\odot \in \{\times, +, \Rightarrow\}$.

Via change of base along $X, Y \longmapsto X \times Y$ one obtains the fibration of (typed) relations $\begin{smallmatrix}\textbf{Rel}\\ \downarrow \\ \textbf{Set}\times\textbf{Set}\end{smallmatrix}$. Objects of **Rel** are relations $S \subseteq U \times V$ and $R \subseteq X \times Y$ and a morphism $S \longrightarrow R$ is a pair of functions $(f : U \longrightarrow X, g : V \longrightarrow Y)$ such that $S(u,v)$ implies $R(f(u), g(v))$. The category **Rel** is Cartesian closed via $\times_{\mathrm{R}}$, $+_{\mathrm{R}}$, and $\Rightarrow_{\mathrm{R}}$. The action on morphisms can be defined with $(u,v) \odot_{\mathrm{R}} (f,g) \;=\; ((u \odot f), (v \odot g))$ for $\odot \in \{\times, +, \Rightarrow\}$ and suitable functions u, v, f and g. See [8] for more details.

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

The remainder of this section presents technical results about the Cartesian structure on predicates and relations. Except from Lemma 1.5 these results are well known and can be found for instance in [8, 13].

Lemma 1.2.

1. *Monotonicity:* $\times_{\mathrm{P}}, +_{\mathrm{P}}, \times_{\mathrm{R}}$, *and* $+_{\mathrm{R}}$ *are monotone (with respect to* $\subseteq$*) in both arguments;* $\Rightarrow_{\mathrm{P}}$ *and* $\Rightarrow_{\mathrm{R}}$ *are antimonotone in the first argument and monotone in the second.*
2. *Truth: The Cartesian closed structure of predicates commutes with truth, that is* $\top_X \odot_{\mathrm{P}} \top_Y = \top_{X \odot Y}$ *for* $\odot \in \{\times, +, \Rightarrow\}$.
3. *Equality: The Cartesian closed structure of relations commutes with equality, that is* $\mathrm{Eq}(A) \odot_{\mathrm{R}} \mathrm{Eq}(B) = \mathrm{Eq}(A \odot B)$ *for* $\odot \in \{\times, +, \Rightarrow\}$.
4. *Opposite Relation: Further we have* $(S \odot_{\mathrm{R}} R)^{op} = S^{op} \odot_{\mathrm{R}} R^{op}$ *for all relations* R *and* S *and* $\odot \in \{\times, +, \Rightarrow\}$.

(5) *Union: For a set A, predicates $P_1, P_2 \subseteq X$, $Q_1, Q_2 \subseteq Y$ and relations $S_1, S_2 \subseteq U \times V$ and $R_1, R_2 \subseteq X \times Y$ we have:*

$$\begin{array}{rcll}
(P_1 \times_{\mathrm{P}} Q_1) \,\cup\, (P_2 \times_{\mathrm{P}} Q_2) & \subseteq & (P_1 \cup P_2) \,\times_{\mathrm{P}}\, (Q_1 \cup Q_2) & (*) \\
(P_1 +_{\mathrm{P}} Q_1) \,\cup\, (P_2 +_{\mathrm{P}} Q_2) & = & (P_1 \cup P_2) \,+_{\mathrm{P}}\, (Q_1 \cup Q_2) & \\
(\top_A \Rightarrow_{\mathrm{P}} Q_1) \,\cup\, (\top_A \Rightarrow_{\mathrm{P}} Q_2) & \subseteq & \top_A \,\Rightarrow_{\mathrm{P}}\, (Q_1 \cup Q_2) & (*) \\
(S_1 \times_{\mathrm{R}} R_1) \,\cup\, (S_2 \times_{\mathrm{R}} R_2) & \subseteq & (S_1 \cup S_2) \,\times_{\mathrm{R}}\, (R_1 \cup R_2) & (*) \\
(S_1 +_{\mathrm{R}} R_1) \,\cup\, (S_2 +_{\mathrm{R}} R_2) & = & (S_1 \cup S_2) \,+_{\mathrm{R}}\, (R_1 \cup R_2) & \\
(\mathrm{Eq}(A) \Rightarrow_{\mathrm{R}} R_1) \,\cup\, (\mathrm{Eq}(A) \Rightarrow_{\mathrm{R}} R_2) & \subseteq & \mathrm{Eq}(A) \,\Rightarrow_{\mathrm{R}}\, (R_1 \cup R_2) & (*)
\end{array}$$

(6) *Intersection: Let $\circledast \in \{\times, +\}$. For a set A, predicates $P_1, P_2 \subseteq X$, $Q_1, Q_2 \subseteq Y$ and relations $S_1, S_2 \subseteq U \times V$ and $R_1, R_2 \subseteq X \times Y$ it holds:*

$$\begin{array}{rcll}
(P_1 \circledast_{\mathrm{P}} Q_1) \,\cap\, (P_2 \circledast_{\mathrm{P}} Q_2) & = & (P_1 \cap P_2) \,\circledast_{\mathrm{P}}\, (Q_1 \cap Q_2) & \\
(\top_A \Rightarrow_{\mathrm{P}} Q_1) \,\cap\, (\top_A \Rightarrow_{\mathrm{P}} Q_2) & = & \top_A \,\Rightarrow_{\mathrm{P}}\, (Q_1 \cap Q_2) & \\
(P_1 \Rightarrow_{\mathrm{P}} Q_1) \,\cap\, (P_2 \Rightarrow_{\mathrm{P}} Q_2) & \subseteq & (P_1 \cap P_2) \,\Rightarrow_{\mathrm{P}}\, (Q_1 \cap Q_2) & (*) \\
(S_1 \times_{\mathrm{R}} R_1) \,\cap\, (S_2 \times_{\mathrm{R}} R_2) & = & (S_1 \cap S_2) \,\times_{\mathrm{R}}\, (R_1 \cap R_2) & \\
(S_1 +_{\mathrm{R}} R_1) \,\cap\, (S_2 +_{\mathrm{R}} R_2) & = & (S_1 \cap S_2) \,+_{\mathrm{R}}\, (R_1 \cap R_2) & (\dagger) \\
(\mathrm{Eq}(A) \Rightarrow_{\mathrm{R}} R_1) \,\cap\, (\mathrm{Eq}(A) \Rightarrow_{\mathrm{R}} R_2) & = & \mathrm{Eq}(A) \,\Rightarrow_{\mathrm{R}}\, (R_1 \cap R_2) & \\
(S_1 \Rightarrow_{\mathrm{R}} R_1) \,\cap\, (S_2 \Rightarrow_{\mathrm{R}} R_2) & \subseteq & (S_1 \cap S_2) \,\Rightarrow_{\mathrm{R}}\, (R_1 \cap R_2) & (*)
\end{array}$$

(7) *Composition: Let again $\circledast \in \{\times, +\}$. 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:*

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

Remark 1.3. 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 (5) (union) does not contain a statement about the general exponent. The two expressions in question are not comparable at all in general. Item (5) (union) and Item (6) (intersection) can be generalized to arbitrary collections, except for line (†), which holds only for nonempty collections.

Proof. All results have been proved within PVS. They are obtained by easy computations. The second equation of (7) (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 role for some results in Section 4. 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 predicates from Remark 1.1 is fibred over an endofunctor K on **Set** if for all $P \subseteq X$ it is the case that $H(P) \subseteq K(X)$. Additionally, for all functions $f : X' \longrightarrow X$ the equation $H(f^* P) = K(f)^* H(P)$ must hold. If H and K are contravariant, then the required equation is $H(\coprod_f P) = (K(f))^* \ H(P)$.

The next result is well known and can be found for instance in [8]. I specialize it to the setting of this paper.

Lemma 1.4. *The Cartesian structure on both **Rel** and **Pred** is fibred.*

(1) *For all predicates $P \subseteq X, Q \subseteq Y$ and functions $f : X' \longrightarrow X$, $g : Y' \longrightarrow Y$, and $g' : Y \longrightarrow Y'$ the following equations hold.*

$$\begin{aligned} (g^* Q) \ \times_{\mathrm{P}} (f^* P) &= (g \times f)^* (Q \times_{\mathrm{P}} P) \\ (g^* Q) \ +_{\mathrm{P}} (f^* P) &= (g + f)^* (Q +_{\mathrm{P}} P) \\ (\textstyle\coprod_{g'} Q) \ \Rightarrow_{\mathrm{P}} (f^* P) &= (g' \Rightarrow f)^* (Q \Rightarrow_{\mathrm{P}} P) \end{aligned}$$

(2) *For all relations $S \subseteq U \times V$ and $R \subseteq X \times Y$ and functions $u : U' \longrightarrow U$, $v : V' \longrightarrow V$, $f : X' \longrightarrow X$, $g : Y' \longrightarrow Y$, and $u' : U \longrightarrow U'$, $v' : V \longrightarrow V'$ it holds that:*

$$\begin{aligned} \big((u \times v)^* S\big) \times_{\mathrm{R}} \big((f \times g)^* R\big) &= ((u \times f) \times (v \times g))^* (S \times_{\mathrm{R}} R) \\ \big((u \times v)^* S\big) +_{\mathrm{R}} \big((f \times g)^* R\big) &= ((u + f) \times (v + g))^* (S +_{\mathrm{R}} R) \\ \big(\textstyle\coprod_{(u' \times v')} S\big) \Rightarrow_{\mathrm{R}} \big((f \times g)^* R\big) &= ((u' \Rightarrow f) \times (v' \Rightarrow g))^* (S \Rightarrow_{\mathrm{R}} R) \end{aligned}$$

Proof. This Lemma has been proved in PVS. The results follow from general properties of the category **Set** and they are immediate, after unpacking the abstract notation. For the exponent of predicates we have for instance

$$\begin{aligned} (\textstyle\coprod_{g'} Q) \ \Rightarrow_{\mathrm{P}} (f^* P) \quad &\subseteq \quad Y' \Rightarrow X' \\ = \quad & \big\{a \mid \forall p' \in Y' . \big(\exists p \in Y . p' = g'(p) \text{ and } Q(p)\big) \ \text{ implies } \ P(f(a(p')))\big\} \\ = \quad & \big\{a \mid \forall p \in Y . Q(p) \ \text{ implies } \ P\big(f(a(g'(p)))\big)\big\} \\ = \quad & (g' \Rightarrow f)^* (Q \Rightarrow_{\mathrm{P}} P) \end{aligned}$$ □

The notion of a cofibred functor is similar to that of a fibred functor: For cofibredness f^* and $\coprod_f$ exchange their role. A functor H on predicates is cofibred over K, if $H(P) \subseteq K(X)$ and $H(\coprod_f P) = \coprod_{K(f)} H(P)$ for a predicate $P \subseteq X$ and a function $f : X \longrightarrow X'$. It is a new observation that it is impossible to get a result like Lemma 1.4 for cofibredness. Although product and coproduct are cofibred for predicates and relations, for the exponent cofibredness fails (for both predicates and relations). The PVS formalization of this paper contains an example for predicates $Q \subseteq Y$, $P \subseteq X$ and functions $f : X \longrightarrow X'$ and $g : Y' \longrightarrow Y$ such that $(g^* Q) \Rightarrow_{\mathrm{P}} (\coprod_f P) \neq \coprod_{g \Rightarrow f} (Q \Rightarrow_{\mathrm{P}} P)$

However, it is possible to recover cofibredness for the exponent in the special case of constant arguments.

Lemma 1.5.

(1) *Let $P \subseteq X$, $Q \subseteq Y$ be predicates and $f : X \longrightarrow X'$, $g : Y \longrightarrow Y'$ be functions. Then*

$$\begin{array}{rcl} \left(\coprod_g Q\right) \times_{\mathrm{P}} \left(\coprod_f P\right) & = & \coprod_{g\times f} (Q \times_{\mathrm{P}} P) \\ \left(\coprod_g Q\right) +_{\mathrm{P}} \left(\coprod_f P\right) & = & \coprod_{g+f} (Q +_{\mathrm{P}} P) \\ \top_Y \Rightarrow_{\mathrm{P}} \left(\coprod_f P\right) & = & \coprod_{\mathrm{id}_Y \Rightarrow f} \left(\top_Y \Rightarrow_{\mathrm{P}} P\right) \end{array}$$

(2) *Assume now relations $S \subseteq U \times V$ and $R \subseteq X \times Y$ and functions $u : U \longrightarrow U'$, $v : V \longrightarrow V'$, $f : X \longrightarrow X'$, $g : Y \longrightarrow Y'$ and let A be an arbitrary set. Then*

$$\begin{array}{rcl} \left(\coprod_{u\times v} S\right) \times_{\mathrm{R}} \left(\coprod_{f\times g} R\right) & = & \coprod_{(u\times f)\times(v\times g)} (S \times_{\mathrm{R}} R) \\ \left(\coprod_{u\times v} S\right) +_{\mathrm{R}} \left(\coprod_{f\times g} R\right) & = & \coprod_{(u+f)\times(v+g)} (S +_{\mathrm{R}} R) \\ \mathrm{Eq}(A) \Rightarrow_{\mathrm{R}} \left(\coprod_{f\times g} R\right) & = & \coprod_{(\mathrm{id}_A \Rightarrow f)\times(\mathrm{id}_A \Rightarrow g)} \left(\mathrm{Eq}(A) \Rightarrow_{\mathrm{R}} R\right) \end{array}$$

Proof. This Lemma has been proved in PVS. The equations for product and coproduct are easy calculations, the third equation of both items requires the Axiom of Choice. Consider for instance the third equation of Item 1: The right hand side is

$$\begin{array}{rl} \coprod_{\mathrm{id}_Y \Rightarrow f} \left(\top_Y \Rightarrow_{\mathrm{P}} P\right) & \subseteq \quad Y \Rightarrow X' \\ & = \quad \{a \mid \exists h : Y \longrightarrow X \,.\, \forall y \in Y \,.\, P(h(y)) \ \text{ and } \ a = f \circ h\} \end{array} \qquad (*)$$

And the left hand side is

$$\top_Y \Rightarrow_{\mathrm{P}} \left(\coprod_f P\right) \quad = \quad \{a \mid \forall y \in Y \,.\, \exists x \in X \,.\, P(x) \ \text{ and } \ a(y) = f(x)\} \qquad (\dagger)$$

There is an obvious inclusion from $(*)$ to $(\dagger)$, for the inclusion in the other direction one has to construct a suitable function h with the Axiom of Choice. □

2. HIGHER-ORDER POLYNOMIAL FUNCTORS

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

Definition 2.1 (Higher-order polynomial functors)**.** A functor $H : \mathbf{Set}^{op} \times \mathbf{Set} \longrightarrow \mathbf{Set}$ is a *higher-order polynomial functor* if it is defined as one of the cases

$$\begin{array}{rcl} H(Y,X) & = & A \mid X \mid H_1(Y,X) \times H_2(Y,X) \mid H_1(Y,X) + H_2(Y,X) \mid \\ & & H_1(X,Y) \Rightarrow H_2(Y,X) \end{array}$$

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 2.2 (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:

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

Because H preserves composition and identities the functional composition of two coalgebra morphism is a coalgebra morphism again. So the above defines a category of H–coalgebras for each higher-order polynomial functor H.

Remark 2.3. 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

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

Example 2.4. 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

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

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:

$$\begin{array}{rclcrcl} \mathsf{get\_x}_c & = & \pi_1 \circ c & \qquad & \mathsf{move}_c & = & \pi_3 \circ c \\ \mathsf{get\_y}_c & = & \pi_2 \circ c & & \mathsf{equal}_c & = & \pi_4 \circ c \\ & & \mathsf{move\_with\_neighbor}_c & = & \pi_5 \circ c & & \end{array}$$

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]

$$\begin{array}{rcl} \mathsf{get\_x}_c(x) & = & \mathsf{get\_x}_d(f(x)) \\ \mathsf{get\_y}_c(x) & = & \mathsf{get\_y}_d(f(x)) \\ f\big(\mathsf{move}_c(x, r_1, r_2)\big) & = & \mathsf{move}_d(f(x), r_1, r_2) \\ \mathsf{equal}_c(x, x') & = & \mathsf{equal}_d(f(x), f(x')) \\ \lambda a \,.\, f\big(\mathsf{move\_with\_neighbor}_c(x, e, r_1, r_2)(a)\big) & = & \\ & & \mathsf{move\_with\_neighbor}_d\big(f(x),\ (\lambda a \,.\, f(e(a))), r_1, r_2\big) \end{array}$$

3. Bisimulations and Invariants for higher-order polynomial functors

Bisimulations are used in various process calculi (for example in [18]) and in the field of coalgebras to capture behavioral equivalence. Invariants[3] are properties that, ones they hold, remain true for all successor states. Greatest invariants contained in some predicate give semantics to the infinitary modal operators always and its dual $\mathsf{eventually}$; see [24] for details. Invariants are also used in coalgebraic refinement [12].

There are two traditions to define the notions of bisimulation and invariant: Following Aczel and Mendler in [1], a bisimulation is the state space of a coalgebra that makes a certain diagram (of coalgebra morphisms) commute. Similarly one can define an invariant as the state space of a subcoalgebra. Rutten, Hennicker and Kurz follow this approach in [25] and [7]. In the following I call this the Aczel/Mendler approach and when necessary I use the term *Aczel/Mendler bisimulation* and *Aczel/Mendler invariant*.

The second tradition stems from Hermida and Jacobs. In [10] they define two special operations for polynomial functors: Predicate and relation lifting. The notions of bisimulation and invariant are then defined with predicate and relation lifting, respectively. This approach is used for instance in [8, 9, 11, 12, 24]. When necessary I use the terms *Hermida/Jacobs bisimulation* and *Hermida/Jacobs invariant* to avoid confusion.

---

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

[3] Invariants are sometimes called subsystems or subcoalgebras.

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

For polynomial functors both approaches are equivalent but they differ for higher-order polynomial functors. In this paper I argue that the Aczel/Mendler approach does not capture the intuitive notions of invariance and of behavioral equivalence for higher-order polynomial functors. 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 3.9 below). Similarly for Aczel/Mendler invariants.

This paper follows therefore Hermida and Jacobs and defines the terms invariant and bisimulation via predicate and relation lifting, respectively.

Definition 3.1 (Predicate and Relation lifting)**.** Let $Q \subseteq Y$ and $P \subseteq X$ be two predicates, $S \subseteq V \times Y$ and $R \subseteq U \times X$ two relations, and H a higher-order polynomial functor. Its predicate lifting $\text{Pred}(H)(Q, P) \subseteq H(Y, X)$ and its relation lifting $\text{Rel}(H)(S, R) \subseteq H(V, U) \times H(Y, X)$ are defined by induction on the structure of H:

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

$$\begin{array}{rcl} \text{Pred}(H)(Q, P) & = & P \\ \text{Rel}(H)(S, R) & = & R \end{array}$$

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

$$\begin{array}{rcccl} \text{Pred}(H)(Q, P) & = & \top_A & = & (A \subseteq A) \\ \text{Rel}(H)(S, R) & = & \text{Eq}(A) & = & \big(\{(a, a) \mid a \in A\} \subseteq A \times A\big) \end{array}$$

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

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

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

$$\begin{array}{rcl} \text{Pred}(H)(Q, P) & = & \text{Pred}(H_1)(Q, P) \ +_{\text{P}} \ \text{Pred}(H_2)(Q, P) \\ & = & \{\kappa_1\, x \mid \text{Pred}(H_1)(Q, P)(x)\} \\ & & \quad \cup \ \{\kappa_2\, y \mid \text{Pred}(H_2)(Q, P)(y)\} \end{array}$$

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

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

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

Predicate and relation lifting for higher-order polynomial functors works on two arguments. In $\mathrm{Pred}(H)(Q,P)$ (respectively $\mathrm{Rel}(H)(S,R)$) the first argument Q (respectively S) is used for the contravariant occurrences of Y in $H(Y,X)$. The second argument P (respectively R) is for the covariant argument of H. The effect is the following: An element $t \in H(Y,X)$ is in $\mathrm{Pred}(H)(Q,P)$ if Q and P hold pointwise on the contravariant and covariant positions in t. For an additional $s \in H(V,U)$ we have $\mathrm{Rel}(H)(S,R)(s,t)$ only if s and t 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 one. 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 predicate and relation lifting is a conservative extension of similar definitions in [10,12]. If H is equivalent to a polynomial functor F, then $\mathrm{Pred}(H)(Q,P) = P^F$ and $\mathrm{Rel}(H)(S,R) = R^F$, where P^F and R^F denote in [12] predicate and relation lifting, respectively. Note, that for a polynomial functor F the contravariant arguments Q and S are ignored. I will therefore write $\mathrm{Pred}(F)(P)$ (respectively $\mathrm{Rel}(F)(R)$) instead of $\mathrm{Pred}(F)(Q,P)$ (respectively $\mathrm{Rel}(F)(S,R)$) for polynomial functors.

I need the following result below.

Lemma 3.2. *Let H be a higher-order polynomial functor.*

(1) *Predicate lifting commutes with truth:*

$$\mathrm{Pred}(H)(\top_Y, \top_X) \quad = \quad \top_{H(Y,X)}$$

(2) *Relation lifting commutes with equality:*

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

(3) *For two relations $S \subseteq U \times V$ and $R \subseteq X \times Y$ relation lifting commutes with $(-)^{op}$ in the following sense:*

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

It is impossible to obtain similar results for the commutation with union, intersection, and relational composition. This fails because it is not possible to obtain equations instead of the subset relations in Lemma 1.2. Predicate and relation lifting for higher-order polynomial functors is also not fibred, because this would require both liftings to be cofibred, which fails for the exponent.

Proof. All three results are obtained by induction on the structure of H, using Lemmas 1.2 (2), 1.2 (3), and 1.2 (4) for the induction steps. ☐

Definition 3.3 (Invariant and 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 predicate $P \subseteq X$ is a H*–invariant* for c if for all $x \in X$

$$P(x) \quad \text{implies} \quad \mathrm{Pred}(H)(P, P)(c(x))$$

- 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 3.4. Assume a point coalgebra $c : X \longrightarrow \mathsf{PointIface}(X, X)$ and a predicate $P \subseteq X$. A function $e : A \longrightarrow X$ restricts to P, if $P(e(a))$ holds for all $a \in A$.[4] The predicate $P \subseteq X$ is a point invariant if, for all $x \in P$, the following two points hold.

- $\forall r_1, r_2 \in \mathbb{R} \,.\, P(\mathsf{move}_c(x, r_1, r_2))$
- the result of $\mathsf{move\_with\_neighbor}_c(x, e, r_1, r_2)$ restricts to P for all $r_1, r_2 \in \mathbb{R}$ and all functions e that restrict to P.

To discuss bisimulations consider an additional coalgebra $d : Y \longrightarrow \mathsf{PointIface}(Y, Y)$ and a relation $R \subseteq X \times Y$. Two functions $e_1 : A \longrightarrow X$ and $e_2 : A \longrightarrow Y$ are R–related if for all $a \in A$ it holds that $R(e_1(a), e_2(a))$.[5] Such a relation R 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 $\mathsf{equal}_c(x, x') \;=_{\mathsf{bool}}\; \mathsf{equal}_d(y, y')$
- For all 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.

---

[4]A function e restricts to P precisely if $\mathrm{Pred}(A \Rightarrow X)(P, P)(e)$ holds.

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

It is now the question if bisimulations and invariants for higher-order polynomial functors enjoy similar properties like for weak pullback preserving endofunctors. 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.

Proposition 3.5. *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.*

(1) *The truth predicate $\top_X$ is an invariant for c.*
(2) *The equality relation $\mathrm{Eq}(X)$ is a bisimulation for c.*
(3) *R^{op} is a bisimulation for d and c if $R \subseteq X \times Y$ is a bisimulation for c and d.*

Proof. Apply Lemma 3.2. □

Observation 3.6. *None* of the following points holds in general for coalgebras of higher-order polynomial functors.

(1) bisimulations and invariants are closed under union,
(2) bisimulations and invariants are closed under intersection,
(3) the composition of two bisimulations is a bisimulation,
(4) the graph and the kernel of a morphism are bisimulations,
(5) the image $\coprod_f \top$ of a morphism f is an invariant,
(6) the predicate $\coprod_{\pi_1} R$ is an invariant for a bisimulation R,
(7) the relation $\coprod_\delta P$ is a bisimulation for an invariant P,[6]
(8) the relation $\pi_1^* P \cap R$ is a bisimulation for an invariant P and a bisimulation R,
(9) and finally invariants correspond to subcoalgebras.

In particular greatest bisimulations and greatest invariants (contained in some predicate) do not exist. So there is no notion of bisimilarity and it is unclear how to define modal operators. I come back to these two problems at the end of Section 4.1 below.

Note that all the points of the preceding observation do hold for coalgebras of polynomial functors. Items (1)–(7) are proved in [25] and Item (8) is proved in [12]. Counterexamples for all items of the preceding observation are contained in the PVS formalization of this paper. It is worth remarking that all the examples for Items (2)–(9) involve the same functor T of the following example. For the union of invariants and bisimulations it is possible to use simpler functors.

Example 3.7. This example constructs two bisimulations such that their intersection is not a bisimulation. Consider the functor

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

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

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

[6] Recall that $\delta = \langle \mathrm{id}, \mathrm{id} \rangle$, so $\coprod_\delta P = \{(x, y) \mid x = y \text{ and } P(x)\}$.

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

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

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:

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

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.

In the remainder of this subsection I define the notions of bisimulation and invariant, following the approach of Aczel and Mendler. It will turn out, that this gives different notions (compared to Definition 3.3). To avoid confusion I use the term *Hermida/Jacobs bisimulation* and *Hermida/Jacobs invariant* for bisimulations and invariants in the sense of Definition 3.3 for the rest of this section.

Definition 3.8 (Aczel/Mendler bisimulations and invariants)**.** Let $c : X \longrightarrow H(X,X)$ and $d : Y \longrightarrow H(Y,Y)$ be coalgebras for a higher-order polynomial functor H.

- A predicate $P \subseteq X$ is called an *Aczel/Mendler invariant* (for c) if there exists a coalgebra $p : P \longrightarrow H(P,P)$ such that the inclusion $\iota : P \hookrightarrow X$ is a H–coalgebra morphism $p \longrightarrow c$.
- A relation $R \subseteq X \times Y$ is called an *Aczel/Mendler bisimulation* c and d) if there exists a coalgebra $r : R \longrightarrow H(R,R)$ such that the projections $\pi_1 : R \longrightarrow X$ and $\pi_2 : R \longrightarrow Y$ are H–coalgebra morphisms.

It is a folklore result that for polynomial functors the notion of Aczel/Mendler bisimulation coincides with Hermida/Jacobs bisimulation and Aczel/Mendler invariant coincides with Hermida/Jacobs invariant. But for higher-order polynomial functors the respective notions are not related in general.

Example 3.9. This example shows a relation R that is an Aczel/Mendler bisimulation for the functor T from Example 3.7 but fails to fulfill Definition 3.3. An example for the converse situation and similar examples for invariants are contained in the PVS formalization of this paper. Take as 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

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

The question is now if R is a bisimulation for c (to instantiate Definition 3.8 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 both $T(\pi_1, M) \circ c$ and $T(\pi_2, M) \circ c$ are constant functions that always return m_1, so r is indeed a coalgebra fulfilling the condition of Definition 3.8.

This shows that the Aczel/Mendler definition for bisimulation does not capture the basic intuition about bisimulations: it is possible to relate states, which are not behaviorally equivalent.

4. 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 Example 3.7 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 [6] nor Eiffel [17] 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++ [26], but there one can use pointers to functions to pass functions as arguments to other functions. However, there are languages that encourage the use of functional arguments. One example is Ocaml [16].

The class Point in the introduction contains an application for arguments of functional type. But even there the type of the argument is Addr$\longrightarrow$**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**$\longrightarrow$**Self**. Therefore I investigate the following restriction of higher-order polynomial functors.

Definition 4.1 (Extended polynomial functors)**.** A functor $G : \mathbf{Set}^{op} \times \mathbf{Set} \longrightarrow \mathbf{Set}$ is called *extended polynomial* if it is built according to the grammar

$$\begin{array}{rcl} G(Y, X) & = & A \ \mid\ X \ \mid\ G_1(Y, X) \times G_2(Y, X) \ \mid\ G_1(Y, X) + G_2(Y, X) \ \mid \\ & & G_1(A, Y) \Rightarrow G_2(Y, X) \end{array}$$

where A is an arbitrary constant set 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 3.7 is not an extended polynomial functor. But the functor $\mathsf{PointIface}$ (Example 2.4) fits into Definition 4.1. 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 2.2), predicate and relation lifting (Definition 3.1), bisimulation, and invariant (Definition 3.3) from higher-order polynomial functors. But because of the restricted exponent these notions behave much more nicely.

In the remainder of this section I prove that all the points listed in Observation 3.6 (2)–(9) (and some additional properties) do hold for extended polynomial functors. For most of the proofs the structure of the proofs and the proof methods are very similar: The property is inferred from a suitable lemma about predicate or relation lifting. These lemmas are proved by induction on the structure of the functor, the induction steps use lemmas of Section 1.

4.1. Intersection and Composition

This subsection shows that invariants and bisimulations for extended polynomial functors are closed under intersection and that bisimulations are closed under composition. I also discuss closure under union.

Lemma 4.2. *Let G be an extended polynomial functor.*

(1) *Let $P_1, P_2 \subseteq X$, $Q_1, Q_2 \subseteq Y$ be predicates. Then*

$$\text{Pred}(G)(Q_1, P_1) \ \cap \ \text{Pred}(G)(Q_2, P_2) \quad \subseteq \quad \text{Pred}(G)(Q_1 \cap Q_2, P_1 \cap P_2)$$

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

$$\text{Rel}(G)(S_1, R_1) \ \cap \ \text{Rel}(G)(S_2, R_2) \quad \subseteq \quad \text{Rel}(G)(S_1 \cap S_2, R_1 \cap R_2)$$

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

$$\text{Rel}(G)(S_1, R_1) \ \circ \ \text{Rel}(G)(S_2, R_2) \quad \subseteq \quad \text{Rel}(G)(S_1 \circ S_2, R_1 \circ R_2)$$

Proof. The proof method that I use is the same for all three items and also for many lemmas in the following. So let me do Item 1 in detail and sketch the rest: First I prove for polynomial functors F by induction on their structure the slightly stronger result

$$\text{Pred}(F)(P_1) \ \cap \ \text{Pred}(F)(P_2) \quad = \quad \text{Pred}(F)(P_1 \cap P_2) \qquad (*)$$

For the induction step I use Lemma 1.2 (6), for instance in case $F = A \Rightarrow F_1$:

$$\begin{array}{rll} \multicolumn{3}{l}{\mathrm{Pred}(A \Rightarrow F_1)(P_1) \;\cap\; \mathrm{Pred}(A \Rightarrow F_1)(P_2)} \\ = & \top_A \Rightarrow_{\mathrm{P}} \mathrm{Pred}(F_1)(P_1) \cap \mathrm{Pred}(F_1)(P_2) & \text{by 1.2 (6)} \\ = & \mathrm{Pred}(A \Rightarrow F_1)(P_1 \cap P_2) & \text{by Ind. Hyp.} \end{array}$$

Now I prove the main result with induction on the structure of extended polynomial functors. In the induction steps I use Lemma 1.2 (6), 1.2 (1), and $(*)$. I demonstrate the case $G = G_1(A, Y) \Rightarrow G_2(Y, X)$. Assume, that the polynomial functor F equals $G_1(A, -)$.

$$\begin{array}{rll} \multicolumn{3}{l}{\mathrm{Pred}(F \Rightarrow G_2)(Q_1, P_1) \;\cap\; \mathrm{Pred}(F \Rightarrow G_2)(Q_2, P_2)} \\ \subseteq & \big(\mathrm{Pred}(F)(Q_1) \cap \mathrm{Pred}(F)(Q_2)\big) \Rightarrow_{\mathrm{P}} & \text{by 1.2 (6)} \\ & \qquad \big(\mathrm{Pred}(G_2)(Q_1, P_1) \;\cap\; \mathrm{Pred}(G_2)(Q_2, P_2)\big) & \\ = & \mathrm{Pred}(F)(Q_1 \cap Q_2) \Rightarrow_{\mathrm{P}} & \text{by } (*) \\ & \qquad \big(\mathrm{Pred}(G_2)(Q_1, P_1) \;\cap\; \mathrm{Pred}(G_2)(Q_2, P_2)\big) & \\ \subseteq & \mathrm{Pred}(F \Rightarrow G_2)(Q_1 \cap Q_2, P_1 \cap P_2) & \text{by Ind. Hyp. and 1.2 (1)} \end{array}$$

This proves Item 1. For Item 2 and 3 I use the same proof method: With Lemma 1.2 I establish first an equality for polynomial functors. Then I prove the main result with induction on the structure of extended polynomial functors. □

Proposition 4.3.

(1) *Invariants for extended polynomial functors are closed under finite intersections.*
(2) *Bisimulations for extended polynomial functors are closed under finite intersection and under composition.*

Proof. Straightforward application of Lemma 4.2. □

Item (1) of the preceding proposition generalizes to intersection over arbitrary collections. So invariants form a complete lattice. However, the join in this complete lattice differs from set-theoretic union and so the greatest invariant contained in some predicate P does not exist in general (because joining all invariants contained in P might yield a predicate greater than P). The PVS formalization contains an example for the functor $(Y, X) \longmapsto Y \Rightarrow X$ that shows this behavior. (Greatest invariants are needed for defining the infinitary modal operators always and $\mathsf{eventually}$, see [24].)

Item (2) does not generalize to arbitrary intersections and bisimulations do not form a complete lattice. In particular there is no greatest bisimulation in general. The PVS formalization of this paper contains an example for the functor $(Y, X) \longmapsto Y \Rightarrow \mathsf{bool}$ with two maximal bisimulations. In the remainder of this subsection I discuss some ideas about what can be done to obtain greatest bisimulations and greatest invariants (contained in some P).

Let us consider bisimulations $R \subseteq X \times X$ for one coalgebra in the following. The domain of such a relation R is defined as $\mathsf{domain}(R) = \coprod_{\pi_1} R \cup \coprod_{\pi_2} R = \{x \mid$

$\exists y \,.\, R(x,y)$ or $R(y,x)\}$. A relation R is *partially reflexive*, if $R(x,y)$ implies both $R(x,x)$ and $R(y,y)$. All partial equivalence relations are partially reflexive.

Proposition 4.4. *Let $c : X \longrightarrow G(X,X)$ be a coalgebra for an extended polynomial functor and let $R, S \subseteq X \times X$ be bisimulations for c. If R and S are partially reflexive relations on the same domain, then there exists a bisimulation that contains both R and S.*

Proof. If R and S are partially reflexive relations with $\mathsf{domain}(R) = \mathsf{domain}(S)$ then $R \cup S \subseteq R \circ S$. Now the result follows from Proposition 4.3 (2). □

Definition 3.3 admits bisimulations that are not partially reflexive and it is even possible to construct a bisimulation R such that there is no bisimulation that contains the partially reflexive closure of R. A restriction to partially reflexive bisimulations seems sensible, because intuitively any state of a system is behavioral equivalent with itself, so any 'sensible' bisimulation should contain the equality relation and thus be partially reflexive. The last proposition implies that there is an upper bound for every two bisimulations that contain the equality relation.

In [20] Poll and Zwanenburg prove that for dialgebras bisimulation equivalences (i.e., bisimulations that are equivalence relations) form a complete lattice. This result can be slightly generalized to a proper subclass of extended polynomial functors. Details will appear elsewhere. One direction for future research is to find a condition that implies the existence of a greatest bisimulation equivalence for all coalgebras of extended polynomial functors. Another problem for future work is the generalization of the notions of partially reflexive relations and that of equivalence relations to obtain similar results for bisimulations $R \subseteq X \times Y$ relating two coalgebras.

Let me discuss now what can be done for greatest invariants. Definition 3.3 is a straightforward generalization of the definition of invariants for polynomial functors. It is not clear if this definition works right in the presence of binary methods. Consider a binary method $\mathsf{m} : \mathsf{Self} \times \mathsf{Self} \longrightarrow \mathsf{Self}$ and assume a predicate $P \subseteq \mathsf{Self}$ and two states $x, y \in \mathsf{Self}$ with $x \in P$ and $y \notin P$. If P is an invariant then $\mathsf{m}(x,x) \in P$ but $\mathsf{m}(x,y)$ might or might not be in P. So an invariant is closed under taking successor states *only* if all arguments for binary methods are taken from the invariant itself.

A *strong invariant* is a predicate P that fulfills (in the context of Definition 3.3)

$$P(x) \quad \text{implies} \quad \mathrm{Pred}(H)(\top_X, P)(c(x))$$

If P is a strong invariant then we have also $\mathsf{m}(x,y) \in P$. Strong invariants are invariants in the sense of Definition 3.3 (because predicate lifting is antimonotone in its first argument). Strong invariants form a complete lattice for all coalgebras of extended polynomial functors, so they can be used to define the semantics of infinitary modal operators.

On the one side strong invariants seem to be more appropriate for some class interfaces with binary methods, on the other side strong invariants do not fit well together with bisimulations (only Proposition 4.11 and 4.13 (1) from Section 4.4 below hold for strong invariants).

4.2. Fibredness and Cofibredness

This subsection investigates fibredness and cofibredness properties of predicate and relation lifting.

Lemma 4.5. *For every polynomial functor F predicate and relation lifting is cofibred.*

(1) *Let $P \subseteq X$ be a predicate and $f : X \longrightarrow X'$ be a function, then*

$$\mathrm{Pred}(F)(\textstyle\coprod_f P) \quad = \quad \textstyle\coprod_{F(f)} \mathrm{Pred}(F)(P)$$

(2) *Let $R \subseteq X \times Y$ be a relation and let $f : X \longrightarrow X'$ and $g : Y \longrightarrow Y'$ be functions, then*

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

Predicate and relation lifting for polynomial functors is also fibred, this follows from the next lemma.

Proof. By induction on the structure of polynomial functors using Lemma 1.5. □

Lemma 4.6. *For all extended polynomial functors G the predicate and relation lifting is fibred.*

(1) *Let $P \subseteq X$, $Q \subseteq Y$ be predicates and $f : X' \longrightarrow X$, $g : Y \longrightarrow Y'$ be functions. Then*

$$\mathrm{Pred}(G)(\textstyle\coprod_g Q, f^* P) \quad = \quad G(g, f)^* \; \mathrm{Pred}(G)(Q, P)$$

(2) *Let $S \subseteq U \times V$, $R \subseteq X \times Y$ be relations and assume four functions $u : U \longrightarrow U'$, $v : V \longrightarrow V'$, $f : X' \longrightarrow X$, $g : Y' \longrightarrow Y$, then*

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

Proof. By induction on the structure of extended polynomial functors using Lemma 1.4 and, for the exponent, Lemma 4.5. □

4.3. Bisimulations and Coalgebra Morphisms

The preceding result about fibredness makes it possible to reuse a folklore proof for the equivalence of (functional) bisimulations and coalgebra morphisms.

Proposition 4.7. *Let G be an extended polynomial functor and assume two G–coalgebras $c : X \longrightarrow G(X, X)$ and $d : Y \longrightarrow G(Y, Y)$. 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 note, that in **Set** there is an equivalence

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

Using this intermediate result I compute

$$\begin{array}{lll} \mathrm{Rel}(G)(\mathsf{graph}(f), \mathsf{graph}(f)) & & \\ \quad = \mathrm{Rel}(G)(\coprod_{\mathrm{id}_X \times f} \mathrm{Eq}(X), (f \times \mathrm{id}_Y)^* \,\mathrm{Eq}(Y)) & & \text{by } (*) \\ \quad = (G(\mathrm{id}_X, f) \times G(f, \mathrm{id}_Y))^* \;\mathrm{Rel}(G)(\mathrm{Eq}(X), \mathrm{Eq}(Y)) & & \text{by 4.6 (2)} \\ \quad = (G(\mathrm{id}_X, f) \times G(f, \mathrm{id}_Y))^* \;\mathrm{Eq}(G(X, Y)) & & \text{by 3.2 (2)} \quad \square \end{array}$$

The kernel of a function $f : X \longrightarrow Y$ is the binary relation on X that 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 4.7 and 4.3 (2).

Proposition 4.8. *The kernel of a morphism of a G–coalgebra is a bisimulation for every extended polynomial functor G.* $\square$

4.4. Bisimulations and Invariants

It is often useful to combine bisimulations and invariants to obtain new bisimulations and/or invariants. The following propositions justify some common constructions.

Lemma 4.9. *Let G be an extended polynomial functor, S and R be arbitrary relations, and P and Q arbitrary predicates. Then*

$$\begin{array}{rcll} \coprod_{\pi_1} \mathrm{Rel}(G)(S, R) & \subseteq & \mathrm{Pred}(G)(\coprod_{\pi_1} S, \; \coprod_{\pi_1} R) & (1) \\ \mathrm{Rel}(G)(S, R) \;\cap\; \pi_1^* \left(\mathrm{Pred}(G)(Q, P)\right) & \subseteq & \mathrm{Rel}(G)(S \cap \pi_1^* (Q), \; R \cap \pi_1^* (P)) & (2) \end{array}$$

Proof. First, one proves for all polynomial functors F by induction on their structure

$$\begin{array}{rcl} \coprod_{\pi_1} \mathrm{Rel}(F)(R) & = & \mathrm{Pred}(F)(\coprod_{\pi_1} R) \\ \mathrm{Rel}(F)(R) \;\cap\; \pi_1^* \left(\mathrm{Pred}(F)(P)\right) & = & \mathrm{Rel}(F)(R \cap \pi_1^* (P)) \end{array}$$

With this the main result follows by induction on the structure of G. The induction steps of this proof (and of the proof of the intermediate equations) have been formalized in PVS. $\square$

Proposition 4.10. *Assume two coalgebras $c : X \longrightarrow G(X, X)$ and $d : Y \longrightarrow G(Y, Y)$ for an extended polynomial functor G. If $R \subseteq X \times Y$ is a bisimulation for c and d then the predicate $\coprod_{\pi_1} R \subseteq X$ is an invariant for c.*

Proof. Assume R is a bisimulation for c and d. Define $P \stackrel{\text{def}}{=} \coprod_{\pi_1} R$. I have to show that for all $x \in P$ also $c(x) \in \mathrm{Pred}(G)(P, P)$. For $x \in P$ there exists $y \in Y$ such that $R(x, y)$ and because R is a bisimulation I have $(c(x), d(y)) \in \mathrm{Rel}(G)(R, R)$. With Lemma 4.9 (1) $c(x) \in \mathrm{Pred}(G)(\coprod_{\pi_1} R, \; \coprod_{\pi_1} R)$ follows. $\square$

Proposition 4.11. *Let $c : X \longrightarrow G(X, X)$ and $d : Y \longrightarrow G(Y, Y)$ be two coalgebras for an extended polynomial functor G. Assume that $R \subseteq X \times Y$ is a bisimulation for c and d and that $P \subseteq X$ is an invariant for c. Then the relation $R \cap \pi_1^* P$ is a bisimulation for c and d.*

Proof. Similar to the proof of 4.10 but using 4.9 (2). □

With the previous two propositions and with Proposition 4.7 it is possible to derive more results.

Proposition 4.12. *Let $c : X \longrightarrow G(X, X)$ be a coalgebra of an extended polynomial functor G. A predicate $P \subseteq X$ is an invariant for c if and only if the diagonal on P, the relation $\coprod_\delta P$, is a bisimulation for c.*

Proof. One direction follows from $\mathrm{Eq}(A) \cap \pi_1^* P = \coprod_\delta P$ with 3.5 (2) and 4.11. The other follows from $\coprod_{\pi_1} \coprod_\delta P = P$ and 4.10. □

Proposition 4.13. *Let $c : X \longrightarrow G(X, X)$ and $d : Y \longrightarrow G(Y, Y)$ be two coalgebras for an extended polynomial functor G and let $f : c \longrightarrow d$ be a H–coalgebra morphism.*

(1) *If $P \subseteq X$ is an invariant for c then $\coprod_f P$ is an invariant for d.*
(2) *If $Q \subseteq Y$ is an invariant for d then $f^* Q$ is an invariant for c.*

Proof. The graph of f is given as $\coprod_{\mathrm{id}_X \times f} \mathrm{Eq}(X) = (f \times \mathrm{id}_Y)^* \,\mathrm{Eq}(Y)$. From preceding propositions I get that $\coprod_{\pi_2} \big(\coprod_{(\mathrm{id}_X \times f)} \mathrm{Eq}(X) \;\cap\; \pi_1^* P\big) = \coprod_f P$ is an invariant. This proves Item 1. For Item 2 I obtain that $\coprod_{\pi_1} \big((f \times \mathrm{id}_Y)^* \,\mathrm{Eq}(Y) \;\cap\; \pi_2^* Q\big) = f^* Q$ is an invariant as well. □

4.5. The Aczel/Mendler Approach Revisited

As next problem I come back to the relation of the two approaches to define bisimulation and invariant. Example 3.9 showed only that both notions of bisimulation are different for the functor T. And T is not an extended polynomial functor.

Proposition 4.14. *For all extended polynomial functors G*

(1) *the notions of Aczel/Mendler invariant and Hermida/Jacobs invariant coincide;*
(2) *the notions of Aczel/Mendler bisimulation and Hermida/Jacobs bisimulation coincide.*

Proof. Let me prove Item 2 first. Assume that $c : X \longrightarrow G(X, X)$ and $d : Y \longrightarrow G(Y, Y)$ are two G–coalgebras and that $R \subseteq X \times Y$ is a relation. Consider the following properties

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

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

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. This completes the proof of Item 2. For this proof the predicates $(*)$ and $(\dagger)$ and the induction steps have been formalized in PVS.

Item 1 follows from Item 2. Let $c : X \longrightarrow G(X, X)$ be a G–coalgebra and $P \subseteq X$ be a predicate. Set $R \stackrel{\text{def}}{=} \mathrm{Eq}(X) \cap \pi_1^* P$, then we have $\coprod_{\pi_1} R = P$. Note that the projection π_1 restricts to an isomorphism $\pi_1 : R \longrightarrow P$. Assume P is an Hermida/Jacobs invariant, then by 4.11 R is a Hermida/Jacobs bisimulation. By Item 2 there exists a $r : R \longrightarrow G(R, R)$ such that R is an Aczel/Mendler bisimulation. From r one can obtain $p : P \longrightarrow G(P, P)$ fulfilling Definition 3.8. For the other direction assume that P is an Aczel/Mendler bisimulation. From its witness p we can construct a function r turning R into an Aczel/Mendler bisimulation. With Item 2 and Proposition 4.10 it follows that P is a Hermida/Jacobs invariant. $\square$

4.6. Final Coalgebras for Extended Polynomial Functors

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 [22].

For all bounded functors (including all polynomial functors) a final coalgebra does exist [15]. So it is natural to ask if those extended polynomial functors that are not equivalent to polynomial functors have final coalgebras. Any extended polynomial functor G for which $G(\mathbf{1}, \mathbf{1})$ is isomorphic to $\mathbf{1}$ has a final coalgebra (this applies also to T from Example 3.7).[7] 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.

Proposition 4.15. *The functor $G(Y, X) = Y \Rightarrow \mathsf{bool}$ does not admit a final coalgebra.*

Proof. Let $\mathsf{bool} = \{\mathsf{t}, \mathsf{f}\}$ 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

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

for G. 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. □

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.

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 caused 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 itself is consistent. Second, he can examine the states of the final model and if he does not find any unwanted behavior, he 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 himself that the final model exists. Once he did this, he 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 [11]. But he does not discuss the question, under which conditions the constructed state space is nonempty.

5. CONCLUSION

This paper extends the notion of coalgebra such that coalgebraic specification can handle binary methods from object-oriented programming. It generalizes polynomial functors to extended polynomial functors and to higher-order polynomial functors and defines the notions of coalgebra, coalgebra morphism, bisimulation, and invariant for such functors. Higher-order polynomial functors can model signatures of classes with arbitrary binary methods. Higher-order polynomial functors are too general: Bisimulations and invariants for higher-order polynomial functors have only few and simple properties.

Extended polynomial functors lie in between polynomial functors and higher-order polynomial functors. Coalgebras for extended polynomial functors can model binary methods that occur in practice. Many important properties of bisimulations and invariants from [25] are proved in this paper for coalgebras of extended polynomial functors. Most of the proofs and all examples have been checked with the theorem prover PVS [19]. Extended polynomial functors seem to be a good compromise between expressiveness and usefulness. The results of this paper are used as a basis for the coalgebraic class specification language CCSL [24], which allows to experiment with class specifications that contain binary methods.

Extended polynomial functors can model methods with structured input types and structured output types. The class of (single-sorted) hidden signatures is therefore a proper subclass of the signatures that can be modeled with extended polynomial functors.

With respect to binary methods, the approach presented here is more expressive than the use of algebraic extensions [7]. The use of coalgebras for extended polynomial functors has further the advantage that binary methods can be specified within the coalgebraic framework.

There are many directions for further research: One interesting point is, whether it is possible to derive more results about the union of bisimulations. The result of Poll and Zwanenburg in [20] on the union of bisimulation equivalences in the context of dialgebras can be slightly generalised to a subclass of extended polynomial functors. It is an open question if one can give a sufficient condition for coalgebras of extended polynomial functors that implies a complete lattice of bisimulation equivalences. Further, it is unclear, if and how this result about bisimulation equivalences and my Proposition 4.4 can be generalised to bisimulations relating two different coalgebras. A second question is if one can allow the (finite) powerset construction in extended polynomial functors to model nondeterminism.

A third more informal question is, 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 [12] 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 fourth interesting question is, under which conditions coalgebraic specifications have final models.

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 '99*, volume 19 of *ENTCS*. Elsevier, Amsterdam, 1999.

[5] J. Goguen and G. Malcolm. A hidden agenda. *Theoretical Computer Science*, 245(1):55–101, August 2000.

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

[7] 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.

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

[9] 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.

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

[11] 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.

[12] 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.

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

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

[15] Y. Kawahara and M. Mori. A small final coalgebra theorem. *Theoretical Computer Science*, 233(1–2):129–145, February 2000.

[16] X. Leroy, D. Doligez, J. Garrigue, D. Rémy, and J. Vouillon. *The Objective Caml system, release 3.01*, March 2001. Available at URL http://caml.inria.fr/ocaml/htmlman/.

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

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

[19] 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.

[20] E. Poll and J. Zwanenburg. From algebras and coalgebras to dialgebras. In A. Corradini, M. Lenisa, and U. Montanari, editors, *Coalgebraic Methods in Computer Science '01*, volume 44 of *ENTCS*. Elsevier, Amsterdam, 2001.

[21] H. Reichel. Behavioural validity of conditional equations in abstract data types. In *Contributions to General Algebra 3*. Teubner, 1985. Proceedings of the Vienna Conference, June 21-24, 1984.

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

[23] Grigore Roşu. *Hidden Logic*. PhD thesis, University of California at San Diego, 2000.

[24] J. Rothe, H. Tews, and B. Jacobs. The coalgebraic class specification language CCSL. *Journal of Universal Computer Science*, 7(2):175–193, March 2001.

[25] J. J. M. M. Rutten. Universal coalgebra: a theory of systems. *Theoretical Computer Science*, 249(1):3–80, October 2000.

[26] B. Stroustrup. *The C++ Programming Language: Third Edition*. Addison-Wesley Publishing Co., Reading, Mass., 1997.

[27] H. Tews. Coalgebras for binary methods. In H. Reichel, editor, *Coalgebraic Methods in Computer Science '00*, volume 33 of *ENTCS*. Elsevier, Amsterdam, 2000.

Communicated by (The editor will be set by the publisher).
April 24, 2001.