From Algebras and Coalgebras to Dialgebras

Erik Poll and Jan Zwanenburg

University of Nijmegen
Toernooiveld 1, 6525 ED Nijmegen, The Netherlands

Abstract

This paper investigates the notion of *dialgebra*, which generalises the notions of algebra and coalgebra. We show that many (co)algebraic notions and results can be generalised to dialgebras, and investigate the essential differences between (co)algebras and arbitrary dialgebras.

1 Introduction

An *algebra* is a set X together with some functions that can be used to *construct* elements of X, i.e. functions f_i that have X as output type,

$$f_i : IN_i(X) \to X,$$

with IN_i a polynomial functor. Algebras are widely used in (theoretical) computer science. E.g. think of algebraic datatypes such as lists and trees, or algebraic specifications.

A *coalgebra* is a set X together with some functions that can be used to *observe* elements of X, i.e. functions f_i that have X as input type,

$$f_i : X \to OUT_i(X),$$

with OUT_i a polynomial functor. Coalgebras can be used to describe various kinds of 'dynamical systems', e.g. automata, processes, or (labelled) transition systems [21]. Moreover, elements of coalgebras can viewed as objects in the sense of object-oriented (OO) programming [17], in which case the operations are viewed as *methods*.

For an introduction to – and a comparison between – algebras and coalgebras we refer to [11]. Algebras and coalgebras are dual notions, and are in some intuitive sense 'opposites'. However, this does not mean that algebra and coalgebra do not have certain things in common. Indeed, *the* standard example of an algebraic specification, stacks, also occurs in the literature as an example of a coalgebraic specification!

[1] Email: {erikpoll,janz}@cs.kun.nl

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

This paper investigates the notion of *dialgebra*, a straightforward generalisation of (co)algebra: a dialgebra is a set X together with some functions

$$f_i : IN_i(X) \to OUT_i(X)$$

with IN_i and OUT_i polynomial functors. The name 'dialgebra' is taken from [4]. Clearly all algebras and coalgebras are dialgebras.

An example of a dialgebra that is neither an algebra nor a coalgebra is a type `Set` of sets of natural numbers with operations

$$\begin{aligned}
\texttt{empty} &: \texttt{Set} \\
\texttt{add} &: \texttt{Set} \times \texttt{Nat} \to \texttt{Set} \\
\texttt{elem} &: \texttt{Set} \times \texttt{Nat} \to \texttt{bool} \\
\texttt{union} &: \texttt{Set} \times \texttt{Set} \to \texttt{Set} \\
\texttt{min} &: \texttt{Set} \to 1 + \texttt{Nat} \\
\texttt{split} &: \texttt{Set} \times \texttt{Nat} \to \texttt{Set} \times \texttt{Set}
\end{aligned}$$

Here `min` could for instance return the minimum of a set, if the set is non-empty, and an element of a unit type 1 if the set is empty, and `split(s,n)` could split the set `s` in a pair of sets, one containing the elements of `s` smaller than `n` and the other containing the rest. This is not an algebra, because for instance `split` and `min` are not 'algebraic', nor is it a coalgebra, because for instance `empty` and the binary operation `union` are not 'coalgebraic'.

Many interesting examples of dialgebras that are not (co)algebras can be obtained simply by extending a coalgebra with an operation $init : 1 \to X$ that yields some initial state. Including such an operation is a very natural thing to do for many examples of coalgebras. In particular, this is very natural when considering objects in the sense of OO: here using dialgebras rather than coalgebras makes it possible to account for the *constructors* as well as the *methods* of a class. In addition it becomes possible to account for so-called *binary methods*.

We will show that most (co)algebraic notions can be defined for the more general dialgebraic case, and investigate in how far properties of (co)algebras – in particular properties of invariants and bisimulations – can be generalised to arbitrary dialgebras, to get a better understanding of what the essential differences between algebras, coalgebras, and dialgebras are.

2 Mathematical Preliminaries

Throughout this paper, types will just be sets. Signatures are mappings from types to types, written as type expressions containing a type variable X.

Definition 2.1 A *signature* $\Sigma(X)$ is a type expression, possibly containing the type variable X, of the form

$$\Sigma(X) ::= X \mid C \mid \Sigma_1(X) + \Sigma_2(X) \mid \Sigma_1(X) \times \Sigma_2(X) \mid \Sigma_1(X) \to \Sigma_2(X),$$

where C comes from a collection of base types. Here $A \times B$ is the Cartesian product, with projections $\pi_1 : A \times B \to A$ and $\pi_2 : A \times B \to B$, and $A + B$ the disjoint sum, with injections $\mathsf{inl} : A \to A + B$ and $\mathsf{inr} : B \to A + B$. The functions $[f_1, f_2] : A_1 + A_2 \to B$ for $f_i : A_i \to B$, and $\langle g_1, g_2 \rangle : A \to B_1 \times B_2$ for $g_i : A \to B_i$ are defined as usual.

A *polynomial signature* is a signature of the form

$$F(X) ::= X \mid C \mid F_1(X) + F_2(X) \mid F_1(X) \times F_2(X).$$

N.B. We deliberately exclude constant exponents of the form $C \to F(X)$ as polynomial functors, because we have not been able to prove our most interesting result, Theorem 3.20, if we include these.

Polynomial signatures are functors, and $F(f) : F(A) \to F(B)$ is defined in the usual way for $f : A \to B$. The notions of predicate and relation lifting can be defined not just for polynomial signature but for all signatures: we can define predicate and relation lifting for arbitrary signatures.

Definition 2.2 For predicates P and Q we define the predicates

- $(P \times^{pred} Q)(x) \iff P(\pi_1(x)) \wedge Q(\pi_2(x))$
- $(P +^{pred} Q)(x) \iff (x = \mathsf{inl}(x') \wedge P(x')) \vee (x = \mathsf{inr}(x') \wedge Q(x'))$
- $(P \to^{pred} Q)(f) \iff \forall x \in A.\ P(x) \Rightarrow Q(f(x))$, with A the domain of P.

Definition 2.3 For relations R and S we define the relations

- $R +^{rel} S = \{(\mathsf{inl}(x), \mathsf{inl}(y)) \mid (x, y) \in R\} \cup \{(\mathsf{inr}(x), \mathsf{inr}(y)) \mid (x, y) \in S\}$
- $R \times^{rel} S = \{(x, y) \mid (\pi_1(x), \pi_1(y)) \in R \wedge (\pi_2(x), \pi_2(y)) \in S\}$
- $R \to^{rel} S = \{(f, g) \mid \forall (x, y) \in R.\ (f(x), g(y)) \in S\}$

Definition 2.4 Let $\Sigma(X)$ be an arbitrary signature. For a predicate P on X and a relation $\sim\ \subseteq X \times Y$, the predicate $\Sigma^{pred}(P)$ on $\Sigma(X)$ and the relation $\Sigma^{rel}(\sim) \subseteq \Sigma(X) \times \Sigma(Y)$ are defined, by induction on the structure of $\Sigma(X)$, as

- if $\Sigma(X) = X$ then $\Sigma^{pred}(P) = P$ and $\Sigma^{rel}(\sim) = \sim$,
- if $\Sigma(X) = C$ then $\Sigma^{pred}(P) = True_C$, the constant predicate 'true' on C, and $\Sigma^{rel}(\sim) = Id_C$, the identity relation on C,
- if $\Sigma(X) = \Sigma_1(X) + \Sigma_2(X)$ then $\Sigma^{pred}(P) = \Sigma_1(P) +^{pred} \Sigma_2(P)$ and $\Sigma^{rel}(\sim) = \Sigma_1(\sim) +^{rel} \Sigma_2(\sim)$,
- if $\Sigma(X) = \Sigma_1(X) \times \Sigma_2(X)$ then $\Sigma^{pred}(P) = \Sigma_1(P) \times^{pred} \Sigma_2(P)$ and $\Sigma^{rel}(\sim) = \Sigma_1(\sim) \times^{rel} \Sigma_2(\sim)$,
- if $\Sigma(X) = \Sigma_1(X) \to \Sigma_2(X)$ then $\Sigma^{pred}(P) = \Sigma_1(P) \to^{pred} \Sigma_2(P)$ and $\Sigma^{rel}(\sim) = \Sigma_1(\sim) \to^{rel} \Sigma_2(\sim)$.

Of course, if we identify predicates with subsets, then there is no difference between Σ and Σ^{pred}. The notion of relation lifting is not just used in the coalgebraic literature (e.g. [10]), but is much more widely used, notably for

logical relations in the semantics of typed lambda calculus (see [12] for a comprehensive overview) and to formalise the notion of parametricity (e.g. see [14]).

Lemma 2.5 *Let $\Sigma(X)$ be an arbitrary signature. Then*

(i) $\Sigma^{pred}(True_X) = True_{\Sigma(X)}$

(ii) $\Sigma^{rel}(Id_X) = Id_{\Sigma(X)}$

Proof. Induction on the structure of $\Sigma(X)$. □

For polynomial signatures, there is a close connection between relation and function lifting:

Lemma 2.6 *For any polynomial signature $F(X)$*

$$graph(F(f)) = F^{rel}(graph(f))$$

where $graph(f) \subseteq A \times B$ is the function $f : A \to B$ viewed as a relation.

Proof. Induction on the structure of $F(X)$. □

Lemma 2.7 *Let $F(X)$ be a polynomial signature, and let i range over I. Then*

(i) $P \subseteq Q \Rightarrow F^{pred}(P) \subseteq F^{pred}(Q)$

(ii) $R \subseteq S \Rightarrow F^{rel}(R) \subseteq F^{rel}(S)$

(iii) $\bigcap_i F^{pred}(P_i) = F^{pred}(\bigcap_i P_i)$

(iv) $\bigcap_i F^{rel}(R_i) = F^{rel}(\bigcap_i R_i)$ *if I is not empty.*

(v) $\bigcup_i F^{pred}(P_i) \subseteq F^{pred}(\bigcup_i P_i)$

(vi) $\bigcup_i F^{rel}(R_i) \subseteq F^{rel}(\bigcup_i R_i)$

(vii) $F^{rel}(R; S) = F^{rel}(R); F^{rel}(S)$

Proof. All these can be proved by induction on the structure of $F(X)$. Properties 3. and 4. are also easy consequences of 1. and 2., respectively. □

None of the properties in Lemma 2.7 hold for arbitrary signatures. Note that we have stronger properties for intersection, (iii) and (iv), than for union, (v) and (vi). The properties $\bigcup_i F^{pred}(P_i) = F^{pred}(\bigcup_i P_i)$ and $\bigcup_i F^{rel}(R_i) = F^{rel}(\bigcup_i R_i)$ do not hold for all polynomial signatures $F(X)$ (for counterexamples, take $F(X) = X \times X$), but do hold for some polynomial signatures:

Lemma 2.8 *Let $F(X)$ be a polynomial signature with at most one occurrence of X. Then*

(i) $\bigcup_i F^{pred}(P_i) = F^{pred}(\bigcup_i P_i)$

(ii) $\bigcup_i F^{rel}(R_i) = F^{rel}(\bigcup_i R_i)$

Proof. Induction on the structure of $F(X)$. Of course, for $F(X)$ with no occurrence of X – i.e. $F(X)$ a constant – the property is trivial. □

3 Dialgebras

Definition 3.1 A *dialgebraic signature* is a signature of the form

$$\Sigma(X) = \Sigma_1(X) \times \ldots \times \Sigma_n(X),$$

with each $\Sigma_i(X)$ of the form

$$\Sigma_i(X) = IN_i(X) \to OUT_i(X),$$

with $IN_i(X)$ and $OUT_i(X)$ polynomial signatures. $\Sigma(X)$ is called *algebraic* iff $OUT_i(X) = X$ for all i, and *coalgebraic* iff $IN_i(X) = X$ for all i.

Throughout the remainder of this paper, Σ will be a dialgebraic signature of the form

$$\Sigma(X) = \prod_{i \in I} \Sigma_i = \prod_{i \in I} IN_i(X) \to OUT_i(X).$$

In examples $\Sigma(X)$ will usually be a labelled product rather than unlabelled one; this is just syntactic sugar.

Definition 3.2 A Σ-*dialgebra* is a pair (A, f) consisting of a set A and a function $f \in \Sigma(A)$, i.e. $f = (f_1, \ldots, f_n)$ with $f_i \in \Sigma_i(A) = IN_i(A) \to OUT_i(A)$.

For (co)algebraic signatures, this is just the definition of Σ-(co)algebra. An important difference between dialgebras and (co)algebras is that whereas an algebra with n operations $f_i : F_i(X) \to X$ can be turned into a algebra with the single operation, namely $[f_1, \ldots, f_n] : F_1(X) + \ldots + F_n(X) \to X$, and, similarly, a coalgebra with n operations $f_i : X \to F_i(X)$ can be turned into a coalgebra with just one operation $\langle f_1, \ldots, f_n \rangle$, we can *not* do something similar for dialgebras. A practical consequence is that most definitions and proofs for dialgebras have to be 'point-wise', quantifying over i.

Definition 3.3 Let (A, f) and (B, g) be Σ-dialgebras. A Σ-*homomorphism* $h : (A, f) \to (B, g)$ is a function from A to B that preserves the operations, i.e.

$$OUT_i(h) \circ f_i = g_i \circ IN_i(h)$$

for all i.

For dialgebras that are (co)algebras, this is the standard notion of (co)algebraic homomorphism.

Lemma 3.4 (i) *The identity id_A is a homomorphism from (A, f) to itself.*

(ii) *Homomorphisms are closed under composition, i.e. if $h : (A, f) \to (A', f')$ and $h' : (A', f') \to (A'', f'')$ then $h' \circ h : (A, f) \to (A'', f'')$.*

Proof. Easy. □

Lemma 2.7 listed some properties for polynomial signatures that do not hold for arbitrary signatures. For dialgebraic signatures, we can salvage some of the properties mentioned in Lemma 2.7:

Lemma 3.5 *Let $\Sigma(X)$ be a dialgebraic signature, and let i range over I. Then*

(i) $\bigcap_i \Sigma^{pred}(P_i) \subseteq \Sigma^{pred}(\bigcap_i P_i)$

(ii) $\bigcap_i \Sigma^{rel}(R_i) \subseteq \Sigma^{rel}(\bigcap_i R_i)$ *if I is not empty.*

(iii) $\Sigma^{rel}(R); \Sigma^{rel}(S) \subseteq \Sigma^{rel}(R; S)$

Proof. The proofs are quite straightforward, using Lemma 2.7. We just give the proof of (i) for binary intersection; the others are similar.

$$
\begin{aligned}
& f \in \Sigma(P \cap Q) \\
\iff & \forall j.\ f_j \in \Sigma_j(P \cap Q) \\
\iff & \forall j.\ f_j \in IN_j(P \cap Q) \to OUT_j(P \cap Q) \\
\iff & \forall j.\ \forall x \in IN_j(P \cap Q).\ f_j(x) \in OUT_j(P \cap Q) \\
\iff & \forall j.\ \forall x \in IN_j(P) \cap IN_j(Q).\ f_j(x) \in OUT_j(P) \cap OUT_j(Q) \\
& \quad \text{by Lemma 2.7(iii) (twice)} \\
\Leftarrow & \ \forall j.\ (\forall x \in IN_j(P).\ f_j(x) \in OUT_j(P)) \\
& \quad \wedge (\forall x \in IN_j(Q).\ f_j(x) \in OUT_j(Q)) \\
\iff & \forall j.\ f_j \in IN_j(P) \to OUT_j(P) \wedge f_j \in IN_j(Q) \to OUT_j(Q) \\
\iff & \forall j.\ f_j \in \Sigma_j(P) \cap \Sigma_j(Q) \\
\iff & f \in \Sigma(P) \cap \Sigma(Q)
\end{aligned}
$$

□

3.1 Invariants and sub-dialgebras

The notion of invariant is used in the literature both for algebras and for coalgebras. Intuitively, a predicate is an invariant if all the operations preserve it:

Definition 3.6 A predicate P on A is an *invariant* for a Σ-dialgebra (A, f) iff $f \in \Sigma^{pred}(P)$.

For dialgebras that are (co)algebras, this is the standard notion of (co)algebraic invariant. Given an invariant we can construct a sub-dialgebra:

Definition 3.7 A *sub-dialgebra* of a Σ-dialgebra (A, f) is another Σ-dialgebra (A', f) with $A' \subseteq A$.

For dialgebras that are (co)algebras, this is the standard notion of sub-(co)algebra. For (A', f) to be a sub-dialgebra of (A, f) all the functions in f have to be 'closed' under the subset A' of A.

Lemma 3.8 *Invariants are closed under intersection.*

Proof. Follows immediately from Lemma 3.5: if $f \in \Sigma(P)$ and $f \in \Sigma(Q)$ then $f \in \Sigma(P) \cap \Sigma(Q) \subseteq \Sigma(P \cap Q)$ by Lemma 3.5. □

As a consequence of this lemma, we can define the smallest invariant of a dialgebra as the intersection of all invariants. This strongest invariant expresses

exactly the property of being 'reachable' by the dialgebra operations.

Invariants of dialgebras are not always closed under union:

Example 3.9 Let $T(X) = X \times X \to X$ and consider the T-(di)algebra $(\mathbb{Z}, +)$. The predicates $Neg(x) = x < 0$ and $Pos(x) = x > 0$ on $\mathbb{Z}$ are both invariants, but clearly their union is not, because the sum of a positive and a negative number may be 0.

For coalgebras, however, we do have this property (e.g. see [21,11]):

Lemma 3.10 *For coalgebras, invariants are closed under union.*

A useful consequence of this lemma is that, for a coalgebra, given any property Φ there exist a largest invariant $\underline{\Phi} \subseteq \Phi$, namely the union of all invariants that are subsets of Φ. We can slightly generalise Lemma 3.10. For this we first define

Definition 3.11 $\Sigma(X)$ *has no binary methods* if none of the $IN_i(X)$ has more than one occurrence of X.

Clearly all coalgebras are dialgebras without binary methods. Many interesting examples of dialgebras without binary methods that are not coalgebras can be obtained in the way mentioned earlier, simply by extending a coalgebra with an operation that yields some initial state.

Note that (counter)Example 3.9 involves a binary method. Binary methods are already notorious in the theoretical computer science literature on object oriented (OO) programming; see [2]. Some properties of coalgebras, that do not hold for all dialgebras, do hold for all dialgebras without binary methods, including:

Lemma 3.12 *For dialgebras without binary methods, invariants are closed under union.*

Proof. Let Σ be a signature without binary methods. It suffices to prove that $\bigcup_i \Sigma^{pred}(P_i) \subseteq \Sigma^{pred}(\bigcup_i P_i)$. The crucial property of dialgebraic signature without binary methods needed to prove this is that, since there is at most one occurrence of X in the $IN_i(X)$, $\bigcup_i F^{pred}(IN_i) = F^{pred}(\bigcup_i IN_i)$ by Lemma 2.8:

$$
\begin{aligned}
& f \in \Sigma(P) \cup \Sigma(Q) \\
\iff\ & \forall j.\ f_j \in \Sigma_j(P) \cup \Sigma_j(Q) \\
\iff\ & \forall j.\ f_j \in IN_j(P) \to OUT_j(P) \vee f_j \in IN_j(Q) \to OUT_j(Q) \\
\iff\ & \forall j.\ (\forall x \in IN_j(P).\ f_j(x) \in OUT_j(P)) \\
& \qquad \vee (\forall x \in IN_j(Q).\ f_j(x) \in OUT_j(Q)) \\
\Rightarrow\ & \forall j.\ \forall x \in IN_j(P) \cup IN_j(Q).\ f_j(x) \in OUT_j(P) \cup OUT_j(Q) \\
\iff\ & \forall j.\ \forall x \in IN_j(P \cup Q).\ f_j(x) \in OUT_j(P) \cup OUT_j(Q) \\
& \qquad \text{since } IN_j(P \cup Q) = IN_j(P) \cup IN_j(Q) \text{ by Lemma 2.8} \\
\Rightarrow\ & \forall j.\ \forall x \in IN_j(P \cup Q).\ f_j(x) \in OUT_j(P \cup Q) \\
& \qquad \text{since } OUT_j(P) \cup OUT_j(Q) \subseteq OUT_j(P \cup Q) \text{ by Lemma 2.7}
\end{aligned}
$$

$\iff \forall j.\ f_j \in IN_j(P \cup Q) \to OUT_j(P \cup Q)$
$\iff \forall j.\ f_j \in \Sigma_j(P \cup Q)$
$\iff f \in \Sigma(P \cup Q)$

□

3.2 Bisimulations, (partial) congruences, and quotient-dialgebras

The notion of bisimulation plays an important role in the literature on coalgebras, as does the closely related notion of (partial) congruence in the literature on algebras.

Definition 3.13 A relation $\sim \subseteq A \times B$ is a *bisimulation* between two Σ-dialgebras (A, f) and (B, g) iff $(f, g) \in \Sigma^{rel}(\sim)$. Dialgebras (A, f) and (B, g) are *bisimilar* if there exists a bisimulation between (A, f) and (B, g).

For (co)algebras one has the property that (co)algebra homomorphisms are just functional bisimulations. This property also holds for dialgebras:

Lemma 3.14 *Let (A, f) and (B, g) be Σ-dialgebras and h be a function from A to B. Then $h : A \to B$ is a homomorphism iff graph$(h) \subseteq A \times B$ is a bisimulation.*

Proof. Straightforward, using Lemma 2.6. □

Lemma 3.15 *Bisimulations are closed under intersection and composition.*

Proof. Follows immediately from Lemma 3.5. □

Bisimulations between dialgebras are not always closed under union. For coalgebras this is a basic property (e.g. see [21,11]):

Lemma 3.16 *For coalgebras, bisimulations are closed under union.*

An important consequence of this property of coalgebras is that between any two coalgebras there exists a largest bisimulation, namely the union of all bisimulations. For arbitrary dialgebras we do not have this property.

Lemma 3.17 *For dialgebras without binary methods, bisimulations are closed under union.*

Proof. Similar to Lemma 3.12. □

Of special interest are bisimulations between a dialgebra and itself:

Definition 3.18 A relation $\sim \subseteq A \times A$ is a *(partial) congruence* on a dialgebra (A, f) iff it is a bisimulation between (A, f) and itself, – i.e. $(f, f) \in \Sigma(\sim)$ – and it is a (partial) equivalence relation.

In [21], for coalgebras, what we call a congruence here is called a bisimulation equivalence. Given a (partial) congruence we can construct a quotient-dialgebra:

Definition 3.19 Let $\sim$ be a (partial) congruence on (A, f). Then the *quotient-dialgebra* $([S]_\sim, [f]_\sim)$ is the Σ-dialgebra $([S]_\sim, [f]_\sim)$, where $[S]_\sim$ is the collection of $\sim$-equivalence classes, and $[f]_\sim$ is the family of functions on $\sim$-equivalence classes induced by f.

As mentioned above, bisimulations are not closed under union. Congruences, however, are, in the following sense:

Theorem 3.20 *Let R_j be congruences on the Σ-dialgebra (A, f), for $j \in J$, $J \neq \emptyset$. Then $(\bigcup_j R_j)^*$ is also a congruence relation for (A, f).*

Proof. See the appendix. □

So, for any dialgebra there exists a largest congruence relation, namely (the transitive closure of) the union of all congruences. Intuitively, this is the notion of observational equality for that dialgebra. In Section 3.3 below, we discuss this difference between dialgebras and coalgebras – the existence of largest congruences vs largest bisimulations – in more detail.

The property above does not hold for partial congruences: [2]

Example 3.21 Let $T(X) = X \times X \to \mathbb{Z}$ and consider the T-(di)algebra $(\mathbb{Z}, f)$ where f is the function

$$f(x, y) = \text{if } y < 0 \text{ then } x \text{ else } 23$$

The relations $R = \mathbb{N} \times \mathbb{N}$ and $Id_\mathbb{Z}$ on $\mathbb{Z}$ are both bisimulations, i.e.

$$(x, x') \in R \wedge (y, y') \in R \Longrightarrow f(x, y) = f(x', y')$$
$$(x, x') \in Id \wedge (y, y') \in Id \Longrightarrow f(x, y) = f(x', y')$$

However, neither $R \cup Id$ nor $(R \cup Id)^*$ is a bisimulation, since for instance $(4, 5) \in R \cup Id$ and $(-1, -1) \in R \cup Id$, but $f(4, -1) \neq f(5, -1)$.

3.3 Coalgebras vs dialgebras: the problem with binary methods

Moving from coalgebras to dialgebras we gain something, notably the possibility of having binary methods. The price for this is that some properties are lost, namely

- the existence of final coalgebras,
- the existence of unique largest bisimulation between any two coalgebras.

These two properties are intimately connected, as the largest bisimulation relates precisely those elements that have the same image under the unique homomorphisms to the final coalgebra. The properties are useful because they provide a canonical notion of observational equality between elements of different coalgebras. Two different coalgebras (A, f) and (B, g) with the

[2] The property does hold for partial congruences if these all have the same domain; instead of the transitive and reflexive closure $(\bigcup_j R_j)^*$ one then has to consider the transitive closure $(\bigcup_j R_j)^+$.

same signature Σ can be regarded as different implementations for classes with the same interface. The properties above then provide a canonical notion of equality between objects from these two classes: an object with an internal state $a \in A$ – using implementation (A, f) – is observationally equal to an object with an internal state $b \in B$ – using implementation (B, g) – iff $a \simeq b$, where $a \simeq b$ is the greatest bisimulation between (A, f) and (B, g).

Below a simple (counter)example to illustrate the fundamental problem with the union of bisimulations caused by a binary method:

Example 3.22 Consider $\Sigma(X) = (X \times X \to X) \times (X \to \mathbb{N})$ and the Σ-dialgebras $Z = (\mathbb{Z}, (+, abs))$ and $L = (List, (+\!+, length))$ with $List$ the set of lists over some type, $+\!+$ the concatenation operation, and $abs : \mathbb{Z} \to \mathbb{N}$ the function returning the absolute value.

The relations $\sim_1$ and $\sim_2$,

$$\sim_1 = \{(z, l) \mid z = length(l)\} \quad \sim_2 = \{(z, l) \mid z = -length(l)\}$$

are bisimulations between Z and L. However, $\sim_1 \cup \sim_2$ is not a bisimulation between Z and L; for example, if l is some list with three elements, then

$$-3(\sim_1 \cup \sim_2)l \ \wedge \ 3(\sim_1 \cup \sim_2)l \ \not\Rightarrow \ -3 + 3 \ (\sim_1 \cup \sim_2) \ l +\!+ l \ .$$

The problem is that when the binary method ($+$ or $+\!+$) is used to observe an individual element (of $\mathbb{Z}$ or $List$, respectively), the operation $+$ of Z offers different – more – observations than the operation $+\!+$ of L.

Binary methods are notorious in the literature on the theoretical foundations of object-oriented programming: in the presence of binary methods defining a satisfactory notion of *subtyping* becomes a problem [2]. This problem is closely related to the fact that there is no canonical notion of 'observational equivalence' in the presence of binary methods. Indeed, the coalgebraic definition of subtyping (see [15]) crucially depends on the existence of final coalgebras, or the existence of unique largest bisimulation between any two coalgebras.

4 Dialgebraic Specification

Just like algebras provide the basis for algebraic specification, coalgebras have been used as the basis for coalgebraic specification, notably in the experimental specification language CCSL [8,20]. We now consider a notion of dialgebraic specification, defined in exactly the same way. Because of lack of space, we have to omit many details of definitions and proofs.

For example, an equational dialgebraic specification for the dialgebraic signature of `Set` mentioned in the introduction could include

```
  union(s,empty) = s
      union(s,t) = union(t,s)
elem(add(s,n),m) = (n=m or elem(s,m))
```

$$\begin{array}{l}
\texttt{elem(union(s,t),n) = elem(s,n) or elem(t,n)} \\
\texttt{elem(empty,n) = false} \\
\texttt{min(s) = inl(x)} \iff \texttt{s = empty} \\
\texttt{min(s) = inr(m)} \Rightarrow \texttt{elem(s,m) = true} \\
\texttt{min(s) = inr(m)} \wedge \texttt{elem(s,n) = true} \Rightarrow \texttt{m} \leq \texttt{n} \\
\texttt{union(split(s,m))} = \texttt{s} \\
\vdots
\end{array}$$

A model of this dialgebraic specification would be a dialgebra providing an implementation of all the operations for which these equations hold. However, instead of insisting that models satisfy the equations above, one could also only require that they satisfy the equations above *up to some congruence relation*. This weaker notion of model makes sense because intuitively a congruence relation provides a notion of 'observational equivalence'.

For example, consider a simple implementation of the specification above using lists, in which `union` is implemented as concatenation $+\!\!+$, i.e. some dialgebra $L = (List_{Nat}, (\ldots, +\!\!+, \ldots)$. This implementation does not satisfy

$$\texttt{union(s,t) = union (t,s)}$$

as concatenation is not commutative, but it does satisfy

$$\texttt{union(s,t)} \sim_{perm} \texttt{union (t,s)}$$

where $\sim_{perm}$ is the relation on lists that relates permutations. If $\sim_{perm}$ is a congruence for L, then L can be regarded as a correct implementation of the specification.

To formally introduce a notion of equational specification for dialgebras, we need to introduce some syntax. We fix a dialgebraic signature $\Sigma(X) = \prod_{i \in I} IN_i(X) \to OUT_i(X)$ and use names op_i for the operations of the dialgebra:

Definition 4.1 The *type expressions* are given by

$$E ::= X \mid C \mid E_1 + E_2 \mid E_1 \times E_2 \mid E_1 \to E_2$$

where X stands for the carrier of the dialgebra, and C ranges over some collection of base types.

The *term expressions* is given by

$$e ::= op_i \mid x^E \mid c^E \mid \lambda x^E.\, e \mid e(e) \mid \mathsf{inl}_{E+E}(e) \mid \mathsf{inr}_{E+E}(e) \mid \pi_1(e) \mid \pi_2(e)$$

where x^E is a variable of type E, and c^E a constant of type E not containing X. We assume an infinite number of variables for every type E. The collection of well-typed terms is defined in the obvious way.

The *propositions* are given by

$$\Phi ::= \neg\Phi \mid \Phi_1 \wedge \Phi_2 \mid e_1 =_E e_2 \mid \forall x^E.\, \Phi$$

where E is some type expression possibly containing X, and e_1 and e_2 are well-formed expressions of type E.

A *dialgebraic specification* Φ is simply a closed proposition.

Note that the definition above allows more propositions than usually allowed in algebraic specification; for example, it allows universal quantifications over the type $X \to X$.

Definition 4.2 The interpretation of a type E, term e, and proposition Φ in the dialgebra (A, f), $[\![E]\!]_{A,f}$, $[\![e]\!]_{A,f}$, and $[\![\Phi]\!]_{A,f}$, are defined in the obvious way, by induction on the structure, interpreting X as A and $op_i : \Sigma_i(X)$ as $f_i : \Sigma_i(A)$.

Because terms and propositions can contain free variables, we have to define the interpretation of terms and propositions wrt. an environment η, $[\![e]\!]^{\eta}_{A,f}$ and $[\![\Phi]\!]^{\eta}_{A,f}$, where the environment η assigns to every free variable x^E an interpretation in $E(A)$.

Definition 4.3 A dialgebra (A, f) *satisfies* specification Φ, written $(A, f) \models \Phi$, iff $[\![\Phi]\!]_{A,f}$ is true.

Now that we have a syntax, we can define a notion of observational equivalence:

Definition 4.4 Two Σ-dialgebras (A, f) and (B, g) are *observationally equivalent* iff for all closed expressions e of some closed type E (i.e. a type E not containing X) the interpretation of e in (A, f) is equal to its interpretation in (B, g).

As one would expect, bisimilar dialgebras are observationally equivalent:

Theorem 4.5 *Let (A, f) and (B, g) be Σ-dialgebras. If (A, f) and (B, g) are bisimilar then they are observationally equivalent.*

Proof. Let $\sim$ a bisimulation between (A, f) and (B, g). For environments η and ξ we write $\eta \sim \xi$ if $(\rho(x^E), \xi(x^E)) \in E^{rel}(\sim)$ for all x^E in their domains. Then we can prove that $([\![e]\!]^{\eta}_{(A,f)}, [\![e]\!]^{\xi}_{(B,g)}) \in E^{rel}(\sim)$ for all $e : E$ and for all $\eta \sim \xi$, by induction on the derivation of e. □

Behavioural Satisfaction

We now consider a weaker notion of *behavioural* satisfaction of dialgebras satisfying specifications 'up to some congruence relation'. First we define $[\![\Phi]\!]_{A,f,\simeq}$, the interpretation of Φ in the dialgebra (A, f) wrt. a congruence $\simeq$:

Definition 4.6 For $\simeq$ a congruence[3] on the dialgebra (A, f), $[\![\Phi]\!]^{\eta}_{A,f,\simeq}$ is defined as $[\![\Phi]\!]^{\eta}_{A,f}$, except that equality is interpreted as follows:

$$[\![e_1 =_E e_1]\!]^{\eta}_{A,f,\simeq} \;=\; ([\![e_1]\!]^{\eta}_{A,f,\simeq}, [\![e_2]\!]^{\eta}_{A,f,\simeq}) \in E^{rel}(\simeq)$$

[3] One could also take a *partial* congruence, but then the semantics of types would have to be changed with $[\![X]\!] = dom(\simeq)$.

Note that if X does not occur in E, this simply reduces to

$$[\![e_1 =_E e_1]\!]^{\eta}_{A,f,\simeq} \;=\; ([\![e_1]\!]^{\eta}_{A,f,\simeq} = [\![e_2]\!]^{\eta}_{A,f,\simeq})$$

A notion of behavioural satisfaction can now be defined as follows:

Definition 4.7 Let (A, f) be a Σ-dialgebra and $\simeq$ a congruence relation for it. Then (A, f) *satisfies* Φ *with respect to* $\simeq$, written $(A, f) \models_{\simeq} \Phi$, iff $[\![\Phi]\!]_{A,f,\simeq}$ is true.

Theorem 4.8 $(A, f) \models_{\simeq} \Phi$ *iff* $(A, f)/\simeq \models \Phi$

Proof. By induction on the structure of Φ we can prove that $[\![\Phi]\!]_{A,f,\simeq} = [\![\Phi]\!]_{(A,f)/\simeq}$. To prove this we must first prove relations between $[\![E]\!]_{A,f}$ and $[\![E]\!]_{(A,f)/\simeq}$, and $[\![e]\!]_{A,f}$ and $[\![e]\!]_{(A,f)/\simeq}$, namely $[\![E]\!]_{A,f} / \simeq = [\![E]\!]_{(A,f)/\simeq}$ and $[[\![e]\!]_{A,f}]_{E(\simeq)} = [\![e]\!]_{(A,f)/\simeq}$. □

Definitions and results similar to the ones above can be found in the literature for algebraic specifications, e.g. in [1,9,19]. We do not know of any similar definitions or results in the literature on coalgebras or coalgebraic specifications. However, given that the notion of 'observability' plays a much more central role in the coalgebraic setting than in the algebraic setting, we believe that a notion of behavioural satisfaction makes even more sense for coalgebraic specifications than for algebraic ones.

More work would be needed to really exploit the opportunities offered by the notion of behavioural satisfaction when reasoning about specifications. In particular, one would want to establish that any consequences of a specification – in a particular logic – are not just valid for models satisfying the specification, but also for models behaviourally satisfying the specification. For algebras and algebraic specifications this idea is pursued in [1,9,19]. In a type-theoretic setting, this idea is illustrated in [16] and further investigated in [23]; the abstract data types considered here are more general than dialgebras.

5 Related Work

Several ways to combine algebras with coalgebras have been investigated over the past few years.

One way of combining algebra with coalgebra is to consider pairs consisting of an algebra and a coalgebra, sometimes called bi-algebras. This is done in [7] and [3]. Dialgebras are clearly more general than algebra-coalgebra pairs. Using a algebra-coalgebra pair rules out operations $f : IN(X) \to OUT(X)$ where both $IN_i(X) \neq X$ and $OUT_i(X) \neq X$, for example a 'partial' binary operation $f : X \times X \to 1 + X$.

In [22] Tews introduces extended polynomial functors and coalgebras for these extended polynomial functors. This setting allows operations that are not possible in our dialgebra-setting, because a (restricted) use of $\to$ is possible in output types. For example, $g : X \to (C_1 \to X) + C_2$ is a coalgebra

for some extended polynomial functor, but cannot be an operation of any dialgebra. However, whereas our notion of dialgebras subsumes algebras, the setting of [22] does not; this setting is still strictly *co*algebraic and does not allow algebraic operations, not even one as simple as $g : C \to X$. In OO terminology, the setting of [22] allows binary methods but not constructors.

As for our dialgebras, for the coalgebras in [22] bisimulations turn out to be closed under intersection and composition, but not under union. It would be interesting to see if a result similar to Theorem 3.20, i.e. closure under union for congruences, could be proved for extended polynomial functors.

Dialgebras and dialgebraic specifications can be regarded as special cases of the abstract data types and the specifications for abstract data types considered in a type-theoretic setting in [16,23]. Such a type-theoretic setting is also used in [5,6]. The crucial observation to link dialgebras with type theory is that dialgebras – and hence algebras and coalgebras – can be regarded as abstract datatypes. Abstract datatypes can be elegantly described in type theory using so-called *existential types* [13], and the logic for the notion of *parametricity* described in [14] then offers the expected proof rule for these existential types, namely that two implementations of an abstract datatype are equal if there exists a bisimulation between them.

This type-theoretic setting allows much wilder signatures than the dialgebraic signatures considered in this paper. This suggests further generalisations, for example with

- operations with higher-order types, e.g. $\texttt{map} : Set_{Nat} \times (Nat \to Nat) \to Set_{Nat}$, or
- polymorphic operations, i.e. operations with type parameters, e.g. $\texttt{polymorphicMap} : Set_{Nat} \to \forall\alpha.(Nat \to \alpha) \to Set_{\alpha}$.

Finally, one of the referees drew our attention to [18], which introduces a notion of 'nested sketches' that support operations with arbitrarily structured in- and output types and seem more general than our dialgebras.

6 Conclusions

We have shown that the notion of dialgebra is a well-behaved generalisation of the notions of algebra and coalgebra. Dialgebras are more general than both algebras and coalgebras. The coalgebraic setting does for instance not allow 'binary methods', i.e. operations of type $f : X \times X \to X$, or operations returning an initial state i.e. operations of type $f : 1 \to X$. The algebraic setting does not allow operations with complicated return types, e.g. 'partial' operations $f : X \to 1 + X$.

We have shown that many notions used in the fields of algebra and coalgebra are essentially identical, and can already be defined for the more general dialgebraic case: the (co)algebraic notions of homomorphism, invariant, bisim-

ulation, and (partial) congruence can all be extended to dialgebras, preserving many of the essential properties.

We have also shown that dialgebraic specification provide a generalisation of (co)algebraic specification, and indicated how the notion of behavioural satisfaction, used in the field algebraic specification, can be extended to dialgebraic specifications (and hence also to coalgebraic specifications).

Given that (co)algebras are special cases of dialgebras, it is to be expected that some properties are lost when moving from algebras or coalgebras to dialgebras.

Most obviously, we no longer have the existence of initial c.q. final models. However, as far as dialgebraic specifications are concerned losing these properties maybe is not too bad, given that one is usually interested in loose semantics anyway.

In addition to this, useful properties of coalgebras that do not hold for arbitrary dialgebras are closure under union for invariants and bisimulations (Lemmas 3.10 and 3.16). The fact that we do not have these closure properties can be traced back to so-called binary methods, which are already notorious in the literature on object-oriented programming because of the problems they cause with *subtyping* (see [2]). For dialgebras without binary methods we do still have the properties that invariants and bisimulations are closed under union.

For a given functor there exists a canonical notion of 'observational equality' between elements of *different* coalgebras for that functor. An important consequence of the fact that for arbitrary dialgebras bisimulations are not closed under union, is that for a dialgebraic signature such a notion may not exist, as discussed in Section 3.3. However, in the dialgebraic setting we do still have a canonical notion of equality between elements of a *single* dialgebra, thanks to Theorem 3.20.

Future Work

Given the duality between algebras and coalgebras it is surprising that, whereas we did come across properties of coalgebras that do not hold for arbitrary dialgebras (namely closure properties for union, Lemmas 3.10 and 3.16), we did not come across (dual) properties of algebras that do not hold for arbitrary dialgebras. Carefully dualising Lemmas 3.10 and 3.16 might reveal such properties.

Another direction for future work is to further investigate which notions and results from the fields of algebra and coalgebra – notably the well-developed field of algebraic specifications – could be generalised to dialgebras.

Finally, the notion of dialgebra we have introduced is fairly ad-hoc and very syntactic. The motivation behind the definition of dialgebra was that it is the natural 'unification' of the definitions of algebra and coalgebra. It would be interesting to investigate more semantical characterisations of some

notion of dialgebra, and to investigate in how far the restriction to polynomial functors could be relaxed, e.g. allowing the extended polynomial functors of [22].

Acknowledgments
We would like to thank Bart Jacobs and Hendrik Tews for comments on earlier versions of this paper.

7 Appendix: Proof of Theorem 3.20

Lemma 7.1 *Basic properties of the operations $+$ and $\times$ on relations are:*

(i) $(R + S); (R' + S') = (R; R') + (S; S')$
(ii) $(R \times S); (R' \times S') = (R; R') \times (S; S')$
(iii) $R^+ + S^+ = (R + S)^+$
(iv) $R^+ \times S^+ = (R \times S)^+$ *if R and S are partially reflexive.*

Here R^+ denotes the transitive closure of R, and R is partially reflexive *iff* $\forall (x, y) \in R.\ (x, x) \in R \wedge (y, y) \in R$.

Proof. Easy. □

Lemma 7.2 *Let F be a polynomial signature. Then $F(R^+) = F(R)^+$ if R is partially reflexive.*

Proof. Induction on the structure of F, using Lemmas 7.1(iii) and 7.1(iv).□

Lemma 7.3 *Let i and j range over I and J, respectively.*

(i) $\bigcup_i R_i + \bigcup_j S_j = \bigcup_{i,j} R_i + S_j$ *if I and J are not empty.*
(ii) $\bigcup_i R_i \times \bigcup_j S_j = \bigcup_{i,j} R_i \times S_j$

Lemma 7.4 *Let F be a polynomial signature. Let R_j be a equivalence relation on A, for all $j \in J$, $J \neq \emptyset$. Then $F\left(\bigcup_j R_j\right) \subseteq \left(\bigcup_j F(R_j)\right)^+$.*

Proof. Induction on the structure of F.

- $F(X) = C$ or $F(X) = X$: trivial.

- $F(X) = F_1(X) + F_2(X)$:

$$\begin{aligned}
&F_1(R_i) \ + \ F_2(R_j) \\
&= F_1(R_i; Id_A) \ + \ F_2(Id_A; R_j) \\
&= (F_1(R_i); F_1(Id_A)) \ + \ (F_2(Id_A); F_2(R_j)) \quad \text{by Lemma 2.7} \\
&= (F_1(R_i) + F_2(Id_A)) \ ; \ (F_1(Id_A) + F_2(R_j)) \text{ by Lemma 7.1(i)} \\
&\subseteq (F_1(R_i) + F_2(R_i)) \ ; \ (F_1(R_j) + F_2(R_j)) \\
&\qquad\qquad \text{since } Id_A \subseteq R_i, \text{ so } F_k(Id_A) \subseteq F_k(R_i) \\
&= F(R_i) \ ; \ F(R_j)
\end{aligned}$$

so

$$\begin{aligned}
F\left(\bigcup_j R_j\right) &= F_1\left(\bigcup_j R_j\right) \ + \ F_2\left(\bigcup_j R_j\right) \\
&\subseteq \left(\bigcup_j F_1(R_j)\right)^+ \ + \ \left(\bigcup_j F_2(R_j)\right)^+ && \text{by IH} \\
&= \left(\bigcup_j F_1(R_j) \ + \ \bigcup_j F_2(R_j)\right)^+ && \text{by Lemma 7.1(iii)} \\
&= \left(\bigcup_{i,j} F_1(R_i) \ + \ F_2(R_j)\right)^+ && \text{by Lemma 7.3(i)} \\
&\subseteq \left(\bigcup_{i,j} F(R_i) \ ; \ F(R_j)\right)^+ && \text{by the result above} \\
&= \left((\bigcup_j F(R_j)) \ ; \ (\bigcup_j F(R_j))\right)^+ && \text{by Lemma 7.1(i)} \\
&\subseteq \left((\bigcup_j F(R_j))\right)^+ && \text{by definition of } +.
\end{aligned}$$

- $F(X) = F_1(X) \times F_2(X)$: Analogous.

□

Lemma 7.5 (Closure of congruences under union)
Let R_j be congruences on the Σ-dialgebra (A, f), for all $j \in J$, $J \neq \emptyset$. Then $\left(\bigcup_j R_j\right)^+$ is a congruence on (A, f).

Proof. We just do the proof for binary union.

Let R and S be congruences on (A, f), i.e. R and S are equivalence relations, $(f, f) \in \Sigma(R)$, and $(f, f) \in \Sigma(S)$. To prove: $(f, f) \in \Sigma\left((R \cup S)^+\right)$, i.e.

$$(f_i, f_i) \in \Sigma_i((R \cup S)^+) = IN_i\left((R \cup S)^+\right) \to OUT_i\left((R \cup S)^+\right)$$

for all i.

Let $(x, x') \in IN_i\left((R \cup S)^+\right)$. To prove $(f_i(x), f_i(x')) \in OUT_i\left((R \cup S)^+\right)$.

$$\begin{aligned}
IN_i\left((R \cup S)^+\right) &= (IN_i(R \cup S))^+ && \text{by Lemma 7.2} \\
&\subseteq \left((IN_i(R) \cup IN_i(S))^+\right)^+ && \text{by Lemma 7.4} \\
&= (IN_i(R) \cup IN_i(S))^+ && \text{by definition of } +.
\end{aligned}$$

So $(x, x') \in (IN_i(R) \cup IN_i(S))^+$, ie. there exist $x_1 \ldots, x_n$ such that

$$(x, x_1),\ (x_1, x_2),\ \ldots,\ (x_n, x') \in IN_i(R) \cup IN_i(S)\ .$$

Since R and S are bisimulations it then follows that

$$\begin{aligned}(f_i(x), f_i(x_1)),\ (f_i(x_1), f_i(x_2)),\ \ldots,\ (f_i(x_n), f_i(x')) \\ \in OUT_i(R) \cup OUT_i(S) \\ \subseteq OUT_i(R \cup S) \text{ by Lemma 2.7}\end{aligned}$$

and hence

$$\begin{aligned}(f_i(x), f_i(x')) \in (OUT_i(R \cup S))^+ \\ = OUT_i\left((R \cup S)^+\right) \text{ by Lemma 7.2}\end{aligned}$$

□

References

[1] Michel Bidoit, Rolf Hennicker, and Martin Wirsing. Behavioral and abstractor specifications. *Science of Computer Programming*, 25:149–186, 1995.

[2] Kim B. Bruce, Luca Cardelli, Giuseppe Castagna, the Hopkins Objects Group (Jonathan Eifrig, Scott Smith, Valery Trifonov), Gary T. Leavens, and Benjamin Pierce. On Binary Methods. *Theory and Practice of Object Systems*, 1(3):221–242, 1996.

[3] Corina Cîrstea. An algebra-coalgebra framework for system specification. In H. Reichel, editor, *Coalgebraic Methods in Computer Science (CMCS'2000)*, number 19 in ENTCS, pages 81–112. Elsevier, Amsterdam, 2000.

[4] Tatsuya Hagino. *A categorical programming language*. PhD thesis, University of Edinburgh, 1987.

[5] Jo Hannay. Specification Refinement with System F. In J. Flum and M. Rodriguez-Artalejo, editors, *Computer Science Logic (CSL'99)*, volume 1683 of *LNCS*, pages 530–545. Springer-Verlag, September 1999.

[6] Jo Hannay. A Higher Order Simulation Relation for System F. In *FOSSACS'2000*, volume 1784 of *LNCS*, pages 130–145. Springer-Verlag, 2000.

[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*, number 19 in ENTCS, pages 195–212. Elsevier, Amsterdam, 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 (ESOP)*, number 1381 in Lecture Notes in Computer Science, pages 105–121. Springer, 1998.

[9] Martin Hofmann and Donald Sannella. On behavioural abstraction and behavioural satisfaction in higher-order logic. *Theoretical Computer Science*, 167:3–45, 1996.

[10] Bart Jacobs. Invariants, bisimulations and the correctness of coalgebraic refinements. In M. Johnson, editor, *Algebraic Methodology and Software Technology (AMAST'97)*, LNCS, pages 276–291. Springer, 1997.

[11] Bart Jacobs and Jan Rutten. A tutorial on (co)algebras and (co)induction. *EATCS Bulletin*, 62:222–259, June 1997.

[12] John C. Mitchell. *Foundations of Programming Languages*. MIT Press, 1996.

[13] John C. Mitchell and Gordon D. Plotkin. Abstract types have existential type. *ACM Trans. on Prog. Lang. and Syst.*, 10(3):470–502, 1988.

[14] Gordon Plotkin and Martin Abadi. A logic for parametric polymorphism. In *Typed Lambda Calculi and Applications*, volume 664 of *Lecture Notes in Computer Science*, pages 361–375, 1993.

[15] E. Poll. A coalgebraic semantics of subtyping. In H. Reichel, editor, *Coalgebraic Methods in Computer Science (CMCS'2000)*, number 33 in ENTCS, pages 281–299. Elsevier, 2000.

[16] E. Poll and J. Zwanenburg. A logic for abstract data types as existential types. In J.-Y. Girard, editor, *Typed Lambda Calculi and Applications (TLCA'99)*, volume 1581 of *LNCS*, pages 310–324. Spinger, 1999.

[17] Horst Reichel. An approach to object semantics based on terminal co-algebras. *Mathematical Structures in Computer Science*, 5:129–152, 1995.

[18] Horst Reichel. Nested sketches. Technical Report ECS-LFCS-98-401, Edinburgh University, Laboratory for Foundations of Computer Science, 1998.

[19] Horst Reichel. Specification semantics. In E. Astesiano, H.-J. Kreowski, and B. Krieg-Brückner, editors, *Algebraic Foundations of System Specifications*, pages 131–158. Springer, 1998.

[20] J. Rothe, B. Jacobs, and H. Tews. The coalgebraic class specification language CCSL. *Journal of Universal Computer Science*, 2001. To appear.

[21] J. Rutten. Universal coalgebra: a theory of systems. *Theoretical Computer Science*, 249:3–80, 2000.

[22] H. Tews. Coalgebras for binary methods. In H. Reichel, editor, *Coalgebraic Methods in Computer Science (CMCS'2000)*, number 33 in ENTCS. Elsevier, Amsterdam, 2000.

[23] Jan Zwanenburg. *Object-Oriented Concepts and Proof Rules: Formalization in Type Theory and Implementation in Yarrow*. PhD thesis, Eindhoven University of Technology, 1999.