View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Universidade do Minho: RepositoriUM

Models as arrows: the role of dialgebras ⋆

Alexandre Madeira[1,2,3], Manuel A. Martins[2], Luís S. Barbosa[1]

[1] Department of Informatics and CCTC, Minho University
[2] Department of Mathematics, University of Aveiro
[3] Critical Software S.A., Portugal

Abstract. A large number of computational processes can suitably be described as a combination of *construction*, i.e. algebraic, and *observation*, i.e. coalgebraic, structures. This paper suggests *dialgebras* as a generic model in which such structures can be combined and proposes a small calculus of dialgebras including a wrapping combinator and sequential composition. To take good care of invariants in software design, the paper also discusses how dialgebras can be typed by predicates and proves that invariants are preserved through composition. This lays the foundations for a full calculus of invariant proof-obligation discharge for dialgebraic models.

1 Introduction

Metaphors. Probably the most elementary model of a computational process is that of a *function* $f : I \longrightarrow O$, which specifies a transformation rule between two structures I and O. In a (metaphorical) sense, this may be dubbed as the 'engineer's view' of reality: *here is a recipe to build gnus from gnats*. Often, however, reality is not so simple. For example, one may know how to produce 'gnus' from 'gnats' but not in all cases. This is expressed by observing the output of f in a more refined context: O is replaced by $O + \mathbf{1}$ and f is said to be a *partial* function. In other situations one may recognise that there is some *context* information about 'gnats' that, for some reason, should be hidden from input. It may be the case that such information is too extensive to be supplied to f by its user, or that it is shared by other functions as well. It might also be the case that building gnus would eventually modify the environment, thus influencing latter production of more 'gnus'. For U a denotation of such context information, the signature of f becomes $f : I \longrightarrow (O \times U)^U$. In both cases f can be typed as $f : I \longrightarrow T\,O$, for $T = Id + \mathbf{1}$ and $T = (Id \times U)^U$, respectively, where T is a functor, intuitively a type transformer providing a *shape* for the output of f.

⋆ Research partially supported by FCT, the *Fundação Portuguesa para a Ciência e a Tecnologia* through *Centro de Investigação e Desenvolvimento em Matemática e Aplicações* of University of Aveiro and the project MONDRIAN (under the contract **PTDC/EIA-CCO/108302/2008**). A. Madeira is also supported by **SFRH/BDE/33650/2009**, a joint PhD grant by *FCT* and *Critical Software S.A., Portugal*.

A function computed within a context is often referred to as 'state-based', in the sense the word 'state' has in automata theory — the memory which both constrains and is constrained by the execution of actions. In fact, the 'nature' of $f : I \longrightarrow (O \times U)^U$ as a 'state-based function' is made more explicit by rewriting its signature as $f : U \longrightarrow (O \times U)^I$

This, in turn, may suggest an alternative model for computations, which (again in a metaphorical sense) one may dub as the 'natural scientist's view'. Instead of a recipe to build 'gnus' from 'gnats', the simple awareness that *there exist gnus and gnats and that their evolution can be observed.* That *observation* may entail some form of *interference* is well known, even from Physics, and thus the underlying notion of computation is not necessarily a passive one.

The able 'natural scientist' will equip herself with the right 'lens' — that is, a tool to observe with, which necessarily entails a particular shape for observation. Similarly, the engineer will resort to a 'tool box' emphasizing the possibility of at least some (essentially finite) things being not only observed, but actually *built.* In summary,

an *observation structure*:	universe $\xrightarrow{c}$ ◯⌒◯ universe
an *assembly process*:	⋒◫ artifact $\xrightarrow{a}$ artifact

Assembly processes are specified in a similar (but dual) way to *observation structures.* Note that in the picture 'artifact' has replaced 'universe', to stress that one is now dealing with 'culture' (as opposed to 'nature') and, what is far more relevant, that the arrow has been *reversed.* Formally, both 'lenses' and 'toolboxes' are functors. And, therefore, an *observation structure* is a ◯⌒◯-coalgebra, and an *assembly process* is a ⋒◫-algebra.

Algebras and coalgebras for a functor [19] provide abstract models of essentially construction (or *data*-oriented) and observation (or *behaviour*-oriented) computational processes, respectively. Construction *compatibility* and *indistinguishability* under observation emerge as the basic notions of equivalence which, moreover, is characterized in a way which is parametric on the particular 'toolbox' or 'lens' used, respectively. Algebraic compatibility and bisimilarity *acquire a shape*, which is the source of abstraction such models are proud of. Moreover, it is well known that, if 'toolboxs' or 'lens' are 'smooth enough', there exist *canonical* representations of all 'artifacts' or 'behaviours into an initial (respectively, final) algebra (respectively, coalgebra).

Purpose and overview. Both *assembly* and *observation* processes, as discussed above, can be modeled by functions, or more generally, by arrows in a suitable category, between the *universes-of-interest.* In such a context, this paper takes a step further in two moves: first it recovers the notion of a *dialgebra* [16, 22] as a suitable way to combine algebraic and coalgebraic aspects of a computational process in a single arrow; then it generalizes its typing universes from sets

to *predicates* to capture the idea that some desirable properties are to be maintained *invariant* along computation, that is, unharmed across all transactions which are embodied in the system's functionality.

The first move is entailed by the authors recent work on modeling *reconfigurable* systems [13]. Such systems may evolve in time through a number of different stages or modes of operation, to which correspond different configurations of the services made available through its interface. Each stage is implemented by an algebra. The component evolution, on the other hand, is modeled by a coalgebra: a configuration changes in response to a particular event in the system. For example, a component in a sensor network may be unable to restart a particular piece of equipment if in an alarm stage of operation, but not in a normal one. On the other hand, the way it computes the result of sensoring a number of hardware control devices may change from one mode to another (resorting *e.g.*, to different weights to use on the weighted sum of measurements). Dialgebras,

$$\overset{\Cap}{\square\!\square}\ U \xrightarrow{\ d\ } \bigcirc\!\frown\!\bigcirc\ U$$

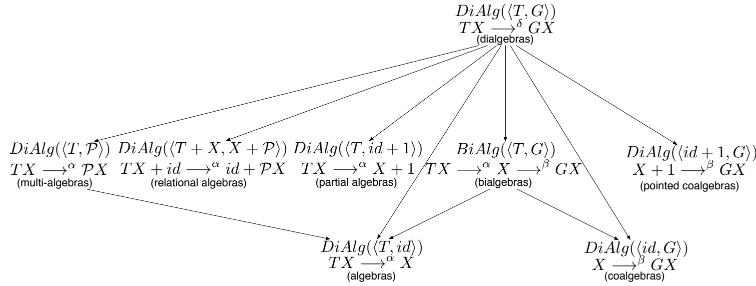
reviewed in section 2, provide, in such cases, a suitable, general model. The second move, on the other hand, types dialgebras by predicates, encoded as coreflexive binary relations. Recall a set X can be represented as a binary relation $y\ \Phi_X\ x \ \equiv\ y = x \ \wedge\ x \in X$, which is called *coreflexive* because it is a fragment of the identity, *i.e.*, $\Phi_X \subseteq id$. The intuition behind this move is that a dialgebra typed by a predicate is a structure for which such a predicate is to be maintained along its evolution, technically, an *invariant*. This emphasizes the role of *invariants* in software design as constraints which restrict behavior in some desirable way, expressing bussiness rules or technical limits, and whose maintenance entails some kind of proof obligation discharge. Section 3 paves the way to a theory of dialgebras which regards invariant predicates as types. An outcome of such a theory is a calculus of invariants' proof obligation discharge, a fragment of which is discussed here, across what is identified as basic algebra of dialgebraic models. The latter consists of two operations — *sequential* composition and *wrapping* — on top of which richer calculi can be developed. Some conclusions and prospects for future work are discussed in section 4.

2 Dialgebras

Categories of dialgebras, were initially defined as *generalized algebraic categories* in [21] and their theory developed in [20, 1]. Later they were studied by [22] in the style of *universal algebra* and of *universal coalgebra* (e.g. [19]). In Computer Science, dialgebras were firstly used in [10] to deal with data types in a purely categorical way. In [17], they are used to specify systems whose states may have an algebraic structure, i.e., as models of evolving algebras of [6]. More recently, dialgebras, as a generalization of both algebras and coalgebras, were studied in [16].

Let us review the basic definition. Let $T, G : \mathbf{C} \to \mathbf{C}$ be two endofunctors over a category $\mathbf{C}$. Formally, a $\langle T, G\rangle$-*dialgebra* consists of a pair $A = \langle A, d\rangle$, where $A \in Obj(\mathbf{C})$ and d is an arrow $d : TA \longrightarrow GA$. A morphism between $\langle T, G\rangle$-dialgebras $A = \langle A, d\rangle$ and $B = \langle B, d'\rangle$ is an arrow $h : A \to B$ such that $d' \cdot Th = G\,h \cdot d$. Dialgebras and their morphisms define a category $DiAlg(\langle T, G\rangle)$.

Dialgebras generalize many interesting structures as depicted below, taking algebras (regarded as $\langle T, id\rangle$-dialgebras) and coalgebras (as $\langle id, G\rangle$-dialgebras) as the simplest instantiations. Let us recall a few examples.



Example 1 (Multi-algebras) *Let $\Sigma = \langle \Lambda, rank\rangle$, where Λ is a set of symbols and rank a function that assigns a natural number to each symbol in Λ, be an algebraic signature, and $T_\Sigma : \mathbf{Set} \to \mathbf{Set}$, defined by $T_\Sigma(X) = \amalg_{\lambda \in \Lambda} X^{rank(\lambda)}$, its associated functor. Objects of $DiAlg(\langle T_\Sigma, \mathcal{P}\rangle)$ are multi-algebras over Σ, ie., pairs $\langle A, \Lambda^A\rangle$ where A is a set and Λ^A is a set of maps $\lambda^A : A^{rank(A)} \to \mathcal{P}(A)$ (cf. [8]). Such structures model nondeterminism of systems, interpreting the operations of the signature as maps that return, for each argument, a set of possible results.*

Example 2 (Partial algebras) *$DiAlg(\langle T_\Sigma, id_{\mathbf{Set}} + 1\rangle)$ is the* category of partial algebras over *Σ. Its objects are pairs $\langle A, \Lambda^A\rangle$ where A is a set and Λ^A a set of partial maps $\lambda^A : A^{rank(\lambda)} \to A$ (cf. [9, Chap 2]). The homomorphisms between two objects $\langle A, \delta\rangle$ and $\langle B, \delta'\rangle$ consists of maps $h : A \to B$ which preserve definability, in the sense that for each $\lambda \in \Lambda$ and for each $rank(\lambda)$-tuple $(a_1, \ldots, a_{rank(\lambda)})$, if $\lambda^A(\langle a_1, \ldots, a_{rank(\lambda)}\rangle) \in A$ then $\lambda^B(\langle h(a_1), \ldots, h(a_{rank(\lambda)})\rangle) \in B$ and $h(\lambda^A(\langle a_1, \ldots, a_{rank(\lambda)}\rangle)) = \lambda^B(\langle h(a_1), \ldots, h(a_{rank(\lambda)})\rangle)$.*

Example 3 (Pointed coalgebras) *A limitation of pure coalgebras to specify initial states of transition systems is overcome in the so called pointed coalgebras, which are objects of $DiAlg(\langle id + 1, G\rangle)$. This makes possible, for example, to introduce initial states on models of automata, as in $d : Q + 1 \longrightarrow Q^{In} \times 2$.*

Example 4 (Bialgebras) *Are pairs $\langle A, (a, c)\rangle$ with $TA \xrightarrow{a} A \xrightarrow{c} GA$, morphisms being simultaneously T-algebra and G-coalgebra morphisms. Clearly, they form a subcategory of $DiAlg(\langle T, G\rangle)$: the corresponding dialgebra is given by $\langle A, c \cdot a\rangle$. Bialgebras were suggested as an adequate structure for software specification in [14], under the name of* algebra-coalgebra pairs *(with respect to (T, G)). Further references [11, 4] studied a specialization of these structures,*

imposing some kind of commitment between the algebraic and coalgebraic components. More precisely, a class of bialgebras was studied for each of which the operations of its algebraic part respect the observations determined by its coalgebraic part.

Reference [13] introduces dialgebras over product types as a model of *reconfigurable systems*. The advantage of such a general way to represent software models is the uniform setting it provides to reason about them. This includes, for example, a notion of *bisimulation*, parametric on functors T and G, and associated calculus. Formally, a *bisimulation* between two dialgebras $d, e \in Obj(DiAlg(\langle T, G \rangle))$ consists of a relation $R \subseteq A \times B$ for which there exists a dialgebraic structure ρ on R making the following diagram to commute.

$$
\begin{array}{ccccc}
TA & \xleftarrow{T\pi_2} & TR & \xrightarrow{T\pi_1} & TB \\
\downarrow{\scriptstyle d} & & \downarrow{\scriptstyle \rho} & & \downarrow{\scriptstyle e} \\
GA & \xleftarrow[G\pi_1]{} & GR & \xrightarrow[G\pi_2]{} & TB
\end{array}
\tag{1}
$$

3 Dialgebras typed by invariants

Invariants. Invariants are constraints on the carrier of dialgebras which restrict their behavior in some desirable way. Formally, an *invariant* for a dialgebra $d : TA \to GA$ is a predicate $P \subseteq A$ satisfying for all $u \in TA$,

$$u \in T(P) \Rightarrow d(u) \in G(P) \tag{2}$$

where $T(P)$, $G(P)$ stands for the *lifting* of predicate P via functors T, G, respectively. Our approach, following previous work in [3], proceeds by transforming (2) into a *pointfree* binary relation formula, *i.e.*, one free of quantifiers and variables (points) such as u above. In this context, we reason:

$$
\begin{array}{ll}
 & \langle \forall\ u\ ::\ u \in T(P)\ \Rightarrow\ d(u) \in G(P) \rangle \\
\equiv & \qquad \{\ \forall\text{-one point rule, } \forall\text{-trading}\ \} \\
 & \langle \forall\ v, u\ ::\ v = u\ \wedge\ u \in T(P)\ \Rightarrow\ d(v) = d(u)\ \wedge\ d(u) \in G(P) \rangle \\
\equiv & \qquad \{\ \text{encoding predicates as coreflexives (twice)}\ \} \\
 & \langle \forall\ v, u\ ::\ v\ \Phi_{T(P)}\ u\ \Rightarrow\ d(v)\ \Phi_{G(P)}\ d(u) \rangle \\
\equiv & \qquad \{\ \text{rule } (f\ b)R(g\ a) \equiv b(f^{\circ} \cdot R \cdot g)a\ \} \\
 & \langle \forall\ v, u\ ::\ v\ \Phi_{T(P)} u\ \Rightarrow\ v(d^{\circ} \cdot \Phi_{G(P)} \cdot d)u \rangle \\
\equiv & \qquad \{\ \text{definition of } \subseteq\ \} \\
 & \Phi_{T(P)}\ \subseteq d^{\circ} \cdot \Phi_{G(P)} \cdot d \\
\equiv & \qquad \{\ \text{law (4) below}\ \} \\
 & c \cdot \Phi_P\ \subseteq F\, \Phi_P \cdot c
\end{array}
\tag{3}
$$

where the last step is justified by the first of the following laws of the relational calculus, known as the *shunting rules* [5], for f a function:

$$f \cdot R \subseteq S \;\equiv\; R \subseteq f^{\circ} \cdot S \qquad \text{and} \qquad R \cdot f^{\circ} \subseteq S \;\equiv\; R \subseteq S \cdot f \tag{4}$$

Note that dialgebra d complies with this rule because it is a function.

Altogether, we arrive at a quite compact definition of what an invariant is, which proves easy to calculate with, as illustrated below. Moreover, the pattern in (3) is an instance of Reynolds "arrow combinator" $R \leftarrow S$ [18] which, given R and S, relates two functions f and g as follows:

$$f(R \leftarrow S)g \quad\equiv\quad f \cdot S \subseteq R \cdot g \tag{5}$$

Instantiating the same pattern with an arbitraty relation R and dialgebras d, e, leads to $d(GR \leftarrow TR)e$ which a simple calculation establishes as an alternative definition of *bisimulation*: as projections π_1, π_2 are functions and inclusion for functions boils down to equality (i.e., $f \subseteq g \;\equiv\; f = g$), diagram (1) can be expressed as $\langle \exists\ \rho \ :: \ d(G\pi_1 \leftarrow T\pi_1)\rho \wedge \rho(G\pi_2 \leftarrow T\pi_2)e\rangle$. Therefore,

$$\begin{array}{cl}
 & \langle \exists\ \rho \ :: \ d(G\pi_1 \leftarrow T\pi_1)\rho \wedge \rho(G\pi_2 \leftarrow T\pi_2)e\rangle \\
\equiv & \qquad \{ \text{ relational converse and composition } \} \\
 & d((G\pi_1 \leftarrow T\pi_1) \cdot (G\pi_2 \leftarrow T\pi_2)^{\circ})e \\
\equiv & \qquad \{ \text{ law } (r \leftarrow f) \cdot (s \leftarrow g)^{\circ} = (r \cdot s^{\circ}) \leftarrow (f \cdot g^{\circ}) \ \} \\
 & d((G\pi_1 \cdot G\pi_2^{\circ}) \leftarrow (T\pi_1 \cdot T\pi_2^{\circ}))e \\
\equiv & \qquad \{ \text{ tabulation: } R = \pi_1 \cdot \pi_2^{\circ} \ \} \\
 & d(GR \leftarrow TR)e
\end{array}$$

The fact that we can write $d(GR \leftarrow TR)e$ instead of $d \cdot TR \subseteq GR \cdot e$ to mean that R is a bisimulation between dialgebras d and e, entails calculational power which is what justifies this recasting bisimulations in terms of Reynolds' arrow combinator. This has been studied in detail in [2], a paper which derives elegant and manageable *pointfree* properties used here.

Calculating invariants. Invariants are, thus, coreflexive bisimulations. Notation $G\Phi_P \xleftarrow{\ d\ } T\Phi_P$ denotes the fact that P is an invariant for d.

This notation suggests a category of "predicates as objects" as a suitable universe for describing dialgebras subject to invariants. Arrows will represent *proof-obligations*. Such a category, although restricted to the coalgebraic case, was studied in detail in [3] to which the interested reader is referred (the generalization to dialgebras is straightforward). The relational characterization of invariants through Reynolds' arrow entails easy to follow, calculational proofs when reasoning about dialgebras. For example, the following result, witnesses

invariant combination and is most useful to deal with the decomposition of the relevant proof-obligations:

$$G\,(\Phi \cdot \Psi) \xleftarrow{d} T(\Phi \cdot \Psi) \quad \Leftarrow \quad G\,\Phi \xleftarrow{d} T\Phi \;\;\wedge\;\; G\,\Psi \xleftarrow{d} T\Psi \tag{6}$$

An algebra of arrows. In general, composition of dialgebras depends on properties of functors T and G. In the sequel two combinators are introduced: *wrapping*, which generalizes interface renaming, and *sequential* composition. These combinators obey a number of laws, such as associativity and a limited form of distributivity, which will not be discussed in this paper. Instead, it will be shown that in all cases *invariants are preserved across composition* — a fundamental healthiness condition for our approach. Actually, such results make possible the structured discharge of the model's proof obligations.

Wrapping. The first combinator is *wrapping*: the interfaces of dialgebra $d : TA \longrightarrow GA$ are extended by pre and post-composition with natural transformations, $\tau_i : T' \Longrightarrow T$ and $\tau_o : G \Longrightarrow G'$. Formally,

$$d[i,o] = \; T'A \xrightarrow{\tau_i} TA \xrightarrow{d} GA \xrightarrow{\tau_o} G'A$$

Lemma 1. *If* $G\,\Phi \xleftarrow{d} T\Phi$ *, then* $G'\Phi \xleftarrow{d[i,o]} T'\Phi$ *.*

Proof.

$$\begin{array}{cl} & \tau_o \cdot d \cdot \tau_i \cdot T'\Phi \\ = & \qquad \{ \text{ naturality of } \tau_i \text{ } \} \\ & \tau_o \cdot d \cdot T\Phi \cdot \tau_i \\ \subseteq & \qquad \{ \text{ hypothesis } \} \\ & \tau_o \cdot G\,\Phi \cdot d \cdot \tau_i \\ = & \qquad \{ \text{ naturality of } \tau_o \text{ } \} \\ & G'\Phi \cdot \tau_o \cdot d \cdot \tau_i \end{array}$$

Pipeline. Sequential composition, *i.e.*, the execution of two dialgebras in pipeline, is only possible in specific situations: basically, with dialgebras in which the algebraic structure is carried to the output side, or, dually, the coalgebraic structure is carried to input. Moreoever, in both cases, additional structure is required from the functors involved. The reason is quite obvious: the formal way to compose arrows whose types appear in a *context* encoded in G (respectively, in F) amounts to lifting the composition diagram to a Kleisli (respectively, co-Kleisli) category of G (resp., T) thought of as a *monad* (resp., *comonad*). Thus, we first distinguish sequential composition for pure algebras and coalgebras.

A monad G, as the reader may recall, encodes computational *effects* (such as exceptions or nondeterminism) in abstract terms, and provides two natural

transformations: the unit $\eta : Id \Longrightarrow G$, which embeds a value in an effect, and a multiplication, $\mu : G \cdot G \Longrightarrow G$, to flatten effects, *i.e.*, to provide a view of a G-effect of a G-effect still as a G-effect. The use of monads to structure the denotational semantics of programming languages was proposed in the 80's, by E. Moggi [15].

Maybe less well-known in semantics, but equally relevant, a comonad [7] entails a dual definition. It models a notion of value in *context*: TA, for T a comonad, is the type of contextually situated values of A. Arrows $m : A \longrightarrow B$ in the corresponding co-Kleisli category are context-relying maps $m : TA \longrightarrow B$ in the base category. Dually to monads, comonads come equipped with a co-unit $\xi : T \Longrightarrow Id$, to extract a value from a context, and a co-multiplication $\nu : T \Longrightarrow T \cdot T$. As expected, η and ξ on the one hand, and μ and ν on the other, satisfy dual laws.

In this setting, sequential composition of coalgebras $c, c' : A \longrightarrow GA$ corresponds to composition in the Kleisli category for G:

$$c \bullet c' = \; A \xrightarrow{c'} GA \xrightarrow{Gc} GGA \xrightarrow{\mu} GA$$

The corresponding role is achieved by co-Kleisli composition for algebras $a, a' : TA \longrightarrow A$:

$$a \star a' = \; TA \xrightarrow{\nu} TTA \xrightarrow{Ta'} TA \xrightarrow{a} A$$

Such definitions apply to dialgebras with the restrictions mentioned above, *i.e.*, whose types are, for arbitrary A, $d : TA \longrightarrow GTA$ (which we call *algebraic-lifted* dialgebras) or $d : TGA \longrightarrow GA$ (called *coalgebraic-lifted*). Invariants are preserved in both cases, *i.e.*,

Lemma 2.

$$G\Phi \xleftarrow{d \star e} TG\Phi \;\Leftarrow\; G\Phi \xleftarrow{d} TG\Phi \;\wedge\; G\Phi \xleftarrow{e} TG\Phi \tag{7}$$

$$GT\Phi \xleftarrow{d \bullet e} T\Phi \;\Leftarrow\; GT\Phi \xleftarrow{d} T\Phi \;\wedge\; GT\Phi \xleftarrow{e} T\Phi \tag{8}$$

Proof. We calculate (7) (the proof of (8) being similar):

$$
\begin{aligned}
& (d \star e) \cdot TG\Phi \\
= \quad & \{ \text{ definition of co-Kleisli composition } \} \\
& e \cdot Td \cdot \nu \cdot TG\Phi \\
= \quad & \{ \ \nu \text{ is a natural transformation } \} \\
& e \cdot Td \cdot TTG\Phi \cdot \nu \\
= \quad & \{ \text{ functoriality } \} \\
& e \cdot TG\Phi \cdot Td \cdot \nu \\
\subseteq \quad & \{ \ G\Phi \xleftarrow{e} TG\Phi \ \text{ and monotonicity } \} \\
& \Phi \cdot e \cdot Td \cdot \nu \\
= \quad & \{ \text{ definition of co-Kleisli composition } \} \\
& \Phi \cdot (e \star d)
\end{aligned}
$$

Clearly, sequential composition for algebras (respectively, coalgebras) comes from (7) (respectively, (8)), taking G (respectively, T) as the identity functor.

4 Conclusions

This paper suggested dialgebras as a suitable model for computational processes combining algebraic and coalgebraic aspects. A small algebra of diagebras was introduced and it is shown that invariants are preserved through composition.

A lot of work remains to be done. Current research focus on the full development of the calculus, including the definition of derived operators, namely for encoding *feedback* mechanisms.

References

1. J. Adámek. Limits and colimits in generalized algebraic categories. *Czechoslovak Mathematical Journal*, 26:55–64, 1976.
2. K. Backhouse and R. Backhouse. Safety of abstract interpretations for free, via logical relations and Galois connections. *SCP*, 15(1–2):153–196, 2004.
3. L. S. Barbosa, J. N. Oliveira, and A. M. Silva. Calculating invariants as coreflexive bisimulations. In J. Meseguer and G. Rosu, editors, *Proc. 12th Inter. Conf. on Algebraic Methodology and Software Technology, AMAST*, pages 83–99. Springer Lect. Notes Comp. Sci. (5140), 2008.
4. M. Bidoit, R. Hennicker, and A. Kurz. Observational logic, constructor-based logic, and their duality. *Theor. Comput. Sci.*, 3(298):471–510, 2003.
5. R. Bird and O. Moor. *The Algebra of Programming.* Series in Computer Science. Prentice Hall, 1997.
6. E. Börger and R. Stärk. *Abstract state machines: A method for high-level system design and analysis.* Springer-Verlag, 2003.

7. S. Brookes and S. Geva. Computational comonads and intensional semantics. In M. P. Fourman, P. T. Johnstone, and A. M. Pitts, editors, *Applications of Categories in Computer Science*, pages 1–44. Cambridge Univ. Press, 1992.
8. A. Corradini and F. Gadducci. Functorial semantics for multi-algebras. In *Recent Trends in Algebraic Development Techniques, volume 1589 of LNCS*, pages 78–90. Springer Verlag, 1998.
9. G. Grätzer. *Universal algebra. 2nd ed.* Springer-Verlag, 1979.
10. T. Hagino. A typed lambda calculus with categorical type constructors. In *Category Theory and Computer Science*, pages 140–157, 1987.
11. R. Hennicker and A. Kurz. (ω, ξ)-logic: On the algebraic extension of coalgebraic specifications. *Electr. Notes Theor. Comput. Sci.*, 19, 1999.
12. A. Kock. Strong functors and monoidal monads. *Archiv für Mathematik*, 23:113–120, 1972.
13. A. Madeira, M. Martins, and L. S. Barbosa. Reconfigurable software components as structured automata. Technical report, Minho University, 2010.
14. G. Malcolm. Behavioural equivalence, bisimulation, and minimal realisation. In *Selected papers from the 11th WS Specification of Abstract Data Types*, pages 359–378. Springer, 1996.
15. E. Moggi. Notions of computation and monads. *Information and Computation*, 93(1):55–92, 1991.
16. E. Poll and J. Zwanenburg. From algebras and coalgebras to dialgebras. In *Coalgebraic Methods in Computer Science '01, volume 44 of ENTCS*, pages 1–19. Elsevier, 2001.
17. H. Reichel. Unifying adt– and evolving algebra specifications. *EATCS Bulletin*, 59:112–126, 1996.
18. J. Reynolds. Types, abstraction and parametric polymorphism. *Information Processing 83*, pages 513–523, 1983.
19. J. Rutten. Universal coalgebra: a theory of systems. *Theoretical Computer Science*, 249:3–80, 2000.
20. V. Trnková. On descriptive classification of set-functors. I. *Commentat. Math. Univ. Carol.*, 12:143–174, 1971.
21. V. Trnková and P. Goralcík. On products in generalized algebraic categories. *Commentationes Mathematicae Universitatis Carolinae*, 1:49–89, 1972.
22. G. Voutsadakis. Universal dialgebra: Unifying universal algebra and coalgebra. *Far East Journal of Mathematical Sciences*, 44(1), 2010.