STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LV**, Number 1, 2010

COALGEBRAIC APPROACH FOR PROGRAM BEHAVIOR IN COMONADS OVER TOPOSES

VILIAM SLODIČÁK, VALERIE NOVITZKÁ

Abstract. The practical goal of program behavior studies is to enhance program and system performance. A behavior of running program can be described by evaluating the coalgebraic structure over a collection of algebraic terms on state space. Coalgebras are defined by polynomial endofunctors. We formulate the coalgebras in categories. Toposes are special kind of categories defined by axioms saying roughly that certain constructions one can make with sets can be done in a category. We use approach via toposes and comonads as dual structures to monads. In our paper we introduce the main ideas of our approach for describing the behavior of systems by coalgebras and we illustrate it on the simple examples of data structure queue.

1. OVERVIEW

The aim of programming is to force the computer to execute some actions and to generate expected behavior. The notion of observation and behavior have played an important *rôle* in computer science. This behavior can be positive, e.g. desired behavior, or negative, e.g. side effect that must be excluded from the system. The description of the behavior of a computer system is non-trivial matter. But some formal description of such complex systems is needed if we wish to reason formally about their behavior. This reasoning should achieve the correctness or security of these systems. The dynamical features of formal structures involve a state of affairs which can be possibly observed and modified. We can consider the computer state as the combined contents of all memory cells. A user is able to observe only a part of this state and he can modify this state by typing commands. As a reaction, the computer displays certain behavior.

Received by the editors: April 10, 2010.

2010 *Mathematics Subject Classification.* 18B25, 18C15.

1998 *CR Categories and Descriptors.* G.2.0 [**Mathematics of computing**]: Discrete mathematics – *General*; F.3.2 [**Logics and meanings of programs**]: Semantics of Programming Languages – *Algebraic approaches to semantics.*

Key words and phrases. Program behavior, Category, Coalgebra, Comonad, Topos.

2. BASIC ASPECTS ABOUT PROGRAM BEHAVIOR

The practical objective of program behavior studies is to enhance program and system performance. The knowledge resulting from these studies may be useful in designing new programs and also may be employed to increase the performance of existing programs and systems [4]. The basic idea of the behavioral theory is to determine the relation between internal states and their observable properties. The internal states are often hidden. Computer scientist have introduced many formal structures to capture the state-based dynamics, e.g. automata, transition systems, Petri nets, or the behavior discovery and verification problem as a graph grammar induction [20]. Horst Reichel firstly introduced the notion of behavior in the algebraic specifications [14]. The basic idea was to divide types in a specification into visible and hidden ones. Hidden types capture states and they are not directly accessible. The execution of a computer program causes generation of some behavior that can be observed typically as computer's input and output [6]. The observation of program behavior can be formularized using coalgebras. A program can be considered as an element of the initial algebra arising from the used programming language. In other words it is an inductively defined set P of terms [11, 16]. This set forms a suitable algebra $F(P) \to P$ where F is an endofunctor constructed over the signature of the operations appointed to execution by a program. Each language construct corresponds to certain dynamics captured in coalgebras. The behavior of programs is described by the final coalgebra $P \to G(P)$ where the functor G captures the kind of behavior that can be observed. Shortly, generated computer behavior amounts to the repeated evaluation of a (coinductively defined) coalgebraic structure on an algebra of terms. Thus coalgebraic behavior is generated by an algebraic program. Therefore the algebras are used for constructing basic structures used in computer programs and coalgebras act on the state space of computer describing what can be observed externally. In our research we are interested into coalgebras because the theory of coalgebras is one of the most promising candidates for a mathematical foundation for computer systems, equipped with both ample applicability and mathematical simplicity. Applicability we conceptualize as mathematical representation of computer system that should be able to serve multifarious theoretical problems in computer systems and their solutions. Mathematical simplicity is a source of abstraction which brings wide applicability.

3. COALGEBRAIC APPROACH

The starting notion in coalgebraic approach is the *signature* used in the theory of algebraic specifications [3]. A signature is a couple $\Sigma = (T, \mathcal{F})$

consisting of a set of types T and a set of function symbols $\mathcal{F}$. In a signature we distinguish:

- *constructor operations* which tell us how to generate (algebraic) data elements;
- *destructor operations*, also called observers or transition functions, that tell us what we can observe about our data elements;
- *derived operations* that can be defined inductively or coinductively.

If we define a derived operation f inductively, we define the value of f on all constructors. In a coinductive definition of f we define the values of all destructors on each outcome $f(x)$.

Example. Let σ be an arbitrary type of queue's elements. We define the signature for the data structure queue as parameterized signature $Queue(\sigma)$:

$$
\begin{array}{ll}
Queue(\sigma): & \\
new: & \to Queue(\sigma) \\
error: & \to \sigma \\
addq: & Queue(\sigma), \sigma \to Queue(\sigma) \\
front: & Queue(\sigma) \to \sigma \\
remove: & Queue(\sigma) \to Queue(\sigma) \\
isEmpty: & Queue(\sigma) \to Bool \\
length: & Queue(\sigma) \to nat \\
if-then-else: & Bool, Queue(\sigma), Queue(\sigma) \to Queue(\sigma)
\end{array}
$$

The operations *new* and *addq* are constructors. Operations *remove* and *front* are destructors. Operations *isEmpty* and *length* are derived operations and they can be defined inductively [11]. □

Operations in a signature determine polynomial endofunctor F that can be constructed inductively from using constants, identities, products, coproducts and exponents. Let $\mathbf{C}$ be a category. An F-algebra

$$a = [cons_1, \ldots, cons_n] : F(\mathbf{C}) \to \mathbf{C}$$

describes internal structure of a program system and consists of co-tuple of constructors [11]. For the program behavior the dual notion of algebra, that is *the coalgebra*, is necessary. A G-coalgeba

$$c = \langle destr_1, \ldots, destr_n \rangle : \mathbf{C} \to G(\mathbf{C}).$$

provides observable properties of a system [7, 11].

For a polynomial endofunctor G a *G-coalgebra* is a pair (U, c), where carrier-set U is a set called *state space* and $c : U \to G(U)$ is the *coalgebraic structure* or

operation of the coalgebra (U, c). It is also known as *coalgebra dynamics*. The difference between F-algebra and G-coalgebra is the same as between construction and observation [11]. While F-algebra tells us how to construct elements in the carrier set by the algebraic structure $a : F(A) \to A$ going *into* A, in the case of coalgebra, the coalgebraic operation $c : U \to G(U)$ goes *out* of U. Usually a program can be considered as an element of the many-typed algebra that arises from used programming language. Each language construct also corresponds to certain dynamics which can be described *via* coalgebras. The program's behavior is thus described by suitable coalgebra acting on the state space of the mathematical machine, i.e. computer [9].

In coalgebras we do not know how to form the elements of U; but we have only the operations working on U, which may give some information about U. Therefore, states can be imagined as a black box and we have only limited access to the state space U. G-coalgebras are also models of the corresponding signatures, but instead of the case of F-algebras, these coalgebras are based on destructor operations.

4. CATEGORIES FOR COALGEBRAS

Coalgebraic concept is based on category theory. The notion of coalgebra is the categorical dual of algebras [2, 5]. Program can be considered as an element of the many-typed algebra that arises from used programming language. Behavior of the program is thus described by suitable coalgebra acting on the state space of the mathematical machine, i.e. computer [9].

4.1. **Coalgebras in category.** Now consider the category of carrier-sets denoted $\mathbf{C}$ with carrier-sets as objects and maps between sets as morphisms. It is a category of sets; according the definition this category is a *topos* [1, 16]. We denote by **Coalg** the category of coalgebras, which we are able construct for any polynomial functor $G : \mathbf{C} \to \mathbf{C}$. For this category we define:

- objects of this category are coalgebras over F, e.g. (U, c), (V, d), ...;
- morphisms are coalgebra homomorphisms between coalgebras, e.g. $f : (U, c) \to (V, d)$, ...;
- every object has identity morphism and coalgebra homomorphisms are composable.

It is well known that in the category **Coalg** of G-coalgebras for a given endofunctor $G : \mathbf{C} \to \mathbf{C}$ the terminal object can be constructed as a limit of a certain descending chain [15]. If this category has terminal object then from every object there is just one morphism to the terminal object. This object is final G-coalgebra that is unique up to isomorphism. A final G-coalgebra

can be obtain from the pure observations. Finality provides a tool for defining functions into final G-coalgebra. A function $f : U \to V$ we define so that we describe direct observations with simple next steps as G-coalgebra on V. A function f arises by steps repeating this process [11].

Example. We can define constructors of signature $Queue(\sigma)$ coalgebraically as follows. Let

$$G(Q) = 1 + (Q \times I)$$

be the polynomial functor. We claim that the final G-coalgebra defined over G is the pair $(I^{\mathbb{N}}, next)$ where map

$$next : I^{\mathbb{N}} \to 1 + (I^{\mathbb{N}} \times I)$$

is defined as

$$next(q) = \begin{cases} \kappa_1(*) & \text{if } q \text{ is empty} \\ \kappa_2(q', i) & \text{if } q = addq(q', i) \end{cases}$$

where κ_1, κ_2 are the first and the second *injections* (co-projections) of the coproduct. The first value of this operation expresses the observation when the queue is empty, i.e. we cannot get any element of the queue. The second value expresses that the operation *next* returns the element (observable value) i of the queue q and the another queue q' without this element, where $q = addq(q', i)$. It holds that

$$q' = remove(q) \qquad\qquad i = front(q)$$

so we obtain

$$q = addq(q', i) = addq\left(remove\left(q\right), front\left(q\right)\right)$$

Then the constructor *new* can be defined as the unique operation $new :\to Queue(\sigma)$ in the commutative diagram at the Fig. 1.

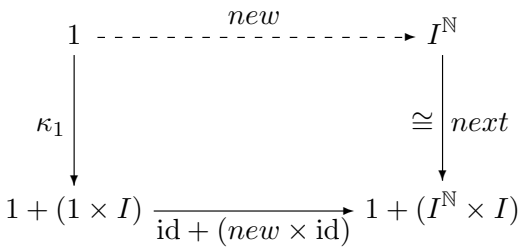

FIGURE 1. Coalgebraic definition of the *new* operator

We are able to define analogously another operations as it was derived in [11, 16].

□

As it was introduced, we can construct category of coalgebras **Coalg** for any polynomial functor $G : \mathbf{C} \to \mathbf{C}$. For a carrier-sets homomorphisms in the category $\mathbf{C}$ we define morphisms between appropriate coalgebras in the category **Coalg**

$$f : (U, c) \to (V, d)$$

such that the diagram at Fig. 2 commutes.

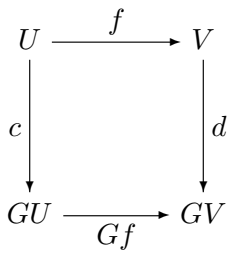

FIGURE 2. Homomorphism of coalgebras and carrier-sets

It follows from the diagram, that there holds the equality

$$Gf \circ c = d \circ f$$

It is clear that the coalgebraic approach is based on categories. The polynomial functors induced by signatures are endofunctors, i.e. their domain and codomain is the same category. The power and expressibility of coalgebraic approach will depend on the concrete categorical structure used for particular programming paradigms [7]. It seems be reasonable that suitable categories will be models of type theories [10, 16]. Their structure and properties ensure the correct treating of data type structures used in all programming paradigms.

It was proved in [2] that the functors

$$\mathcal{U}_G : \mathbf{Coalg} \to \mathbf{C} \qquad \mathcal{F}_G : \mathbf{C} \to \mathbf{Coalg}$$

form a pair of adjoint functors.

Functor $\mathcal{U}_G$ is the forgetful functor and it assigns to each coalgebra its carrier-set and to each coalgebra homomorphism the function between carrier-sets:

$$\mathcal{U}_G\left(\left(U, U \xrightarrow{c} GU\right)\right) = U \qquad \mathcal{U}_G(f) = f$$

Analogously we define generating functor $\mathcal{F}_G$. It assigns to each carrier-set the appropriate coalgebra and to each function between carrier-sets it assigns

coalgebra homomorphism:

$$\mathcal{F}_G(U) = \left(GU, GU \xrightarrow{\delta U} GGU\right) \qquad \mathcal{F}_G(f) = Gf$$

where δ is comonadic operation *co-multiplication* and GGU can be shortly redrawn as G^2U.

Functors $\mathcal{U}_G$ and $\mathcal{F}_G$ form an adjoint pair of functors $\mathcal{F}_G \dashv \mathcal{U}_G$

$$\mathcal{F}_G \dashv \mathcal{U}_G$$

which means that their composition is polynomial functor G

$$\mathcal{U}_G \circ \mathcal{F}_G = G$$

This functor is considered as a functor for comonad over category of carrier-sets [1, 16]. As we defined the pair of adjoint functors $\mathcal{F}_G$ and $\mathcal{U}_G$, we consider the co-unit of comonad ε as the co-unit of adjunction [18, 19].

5. COMONADIC APPROACH AND COALGEBRAS

From one point of view, a monad is an abstraction of certain properties of algebraic structures. From another point of view, it is an abstraction of certain properties of adjoint functors. Theory of monads has turned out to be an important tool for studying toposes [2, 16]. We are interested to explanation of categorical formulation of coalgebras for program behavior. Comonads allow us to utilize the properties of coalgebras in categories. While comonads have very important connection to toposes, we will be concerned about the representation of coalgebras via comonads in toposes.

5.1. **Definition of comonad.** Comonad (or cotriple) is dual construction to monad. The comonad over category $\mathbf{C}$ is the monad over opposite category $\mathbf{C}^{op}$.

Comonad is a mathematical structure $\mathcal{G} = (G, \varepsilon, \delta)$ which consists of

- endofunctor $G : \mathbf{C} \to \mathbf{C}$;
- co-unit natural transformation $\varepsilon : G \to \mathrm{id}_{\mathbf{C}}$;
- co-multiplication natural transformation $\delta : G \to G^2$.

Morphisms ε and δ are natural transformations of the functor G. They are subject to the condition that the diagrams at Fig. 3 and Fig. 4 commute. Coherence triangle at Fig. 4 is the combination of two commutative triangles. It defines left and right identity according to definition of the category. The component of εG at some object X is the component of ε at GX, whereas the component of $G\varepsilon$ at X is $G(\varepsilon X)$ (as at Fig. 6); similar descriptions apply to

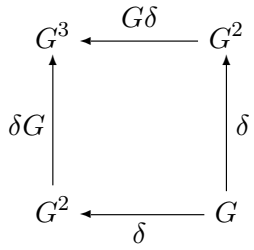

FIGURE 3. Coherence square for comonad

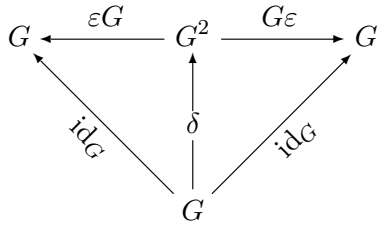

FIGURE 4. Coherence triangle for comonad

δ at Fig. 5.

We say that the comonad is *left-exact* if the functor G is left exact. Functor G is left exact, when it preserves binary products, terminal object and equalizers [2]. For coalgebras we define commutative coherence diagrams over comonad (Fig. 5 and Fig. 6).

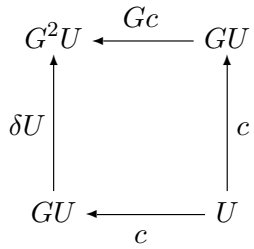

FIGURE 5. Coherence square for coalgebra

Coherence square at Fig. 5 is the basic diagram for generating functor $\mathcal{F}_G$. We are able to construct coalgebra dynamics with that relations for functors (Fig.

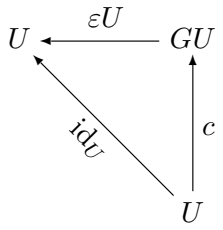

FIGURE 6. Coherence triangle for coalgebra

6). For coalgebras and their state spaces we define relation by the commutative diagram at Fig. 5.
It follows from the diagram at Fig. 5 that the following equation holds:

$$Gf \circ c = d \circ f$$

It means that all paths from the object U into the object GV in diagram constructed as compositions of corresponding arrows are equal in the commutative diagram at Fig. 5.

The diagrams at Fig. 5 and Fig. 6 satisfy the definition of comonadic natural transformations and express how to define coalgebra dynamics and construction of coalgebra dynamics by the generating functor $\mathcal{F}_G$. They also show the definition of comonad by coalgebras.

5.2. **The definition of topos.** Now we formulate basic aspects about toposes. Toposes are important in theoretical informatics for the purpose of modeling computations [1, 13, 17]. A topos is a special kind of category defined by axioms saying roughly that certain constructions one can make with sets can be done in a category [12]. If we want a topos to be a generalized mathematical theory, we suppose that a set of hypotheses or axioms are formulated in predicate logic. They implicitly define some kind of structure of objects and some properties of morphisms in the category $\mathcal{E}$. A topos is really a structure of a general theory defined by axioms formulated possibly in higher-order logic [16]. An elementary topos is such one whose axioms are formulated in the first-order logic, *i.e.* an elementary topos is the generalized axiomatic set theory.

A topos is a special category satisfying important conditions. There are many definitions of topos. In the following text we follow this definition:
A *topos* is a category $\mathcal{E}$ which satisfies the following properties:

- it is cartesian closed category;
- it has finite limits;

- it has representable subobject functor.

5.3. **Coalgebras in toposes.** In [11, 16] we formulated algebras in monadic terms. Analogously we formulated here coalgebras in monads. Kleisli categories are powerful tool for formulating properties of coalgebras over monads. We are able to formulate coalgebras also in categorical terms over comonad. In [16] we showed that toposes as special categories are able to formulate the properties of coalgebras in categorical terms. Category of carrier-sets consists of sets as objects and functions as morphisms between them. While category of sets is a topos [1, 8], the category of carrier-sets is also topos.

We defined coalgebraic operation *new* in chap. 4.1. Now we will consider the codomain of *next* - namely $1 + \left(I^{\mathbb{N}} \times I\right)$ as the subobject classifier Ω. This classifier together with defined morphism *true* will classify whether the subobjects of a given object 1 according to which elements belong to the subobject. We redraw the diagram in Fig. 1 as the diagram in Fig. 7.

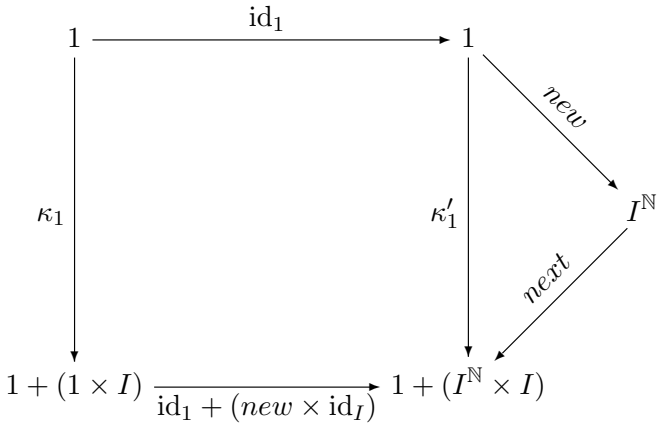

FIGURE 7. Coalgebraic definition of the *new* operator in topos

The morphism $true : 1 \to \Omega$ is here defined as a pair of morphisms into coproduct:

$$true: \quad 1 \underset{next \circ new}{\overset{\kappa'_1}{\rightrightarrows}} \Omega$$

where $\Omega = 1 + (I^{\mathbb{N}} \times I)$. It satisfies the assumption about elements of queue that

$$next(q) = \begin{cases} \kappa_1(*) & \text{if } q \text{ is empty} \\ \kappa_2(q', i) & \text{if } q = addq(q', i) \end{cases}$$

so we are able with subobject classifier to construct coalgebra operations in topos.

Comonad $\mathcal{G} = (G, \varepsilon, \delta)$ in which G is left-exact functor is called *left-exact comonad*. It holds that for topos $\mathcal{E}$ and for the left-exact comonad $\mathcal{G}$ in $\mathcal{E}$ the category **Coalg** of coalgebras of $\mathcal{G}$ is also a topos [2]. From this property we are able to define previous constructions in toposes - we can define the properties of coalgebras in terms of toposes. For example the notion of *subcoalgebra* and the bisimilarity relation [6] we formulate as a subobject [16].

6. Conclusion

In our paper we presented the main ideas of our approach to the behavioral theory. We explained polynomial endofuctor that is the basic concept in algebraic and coalgebraic approach. The concepts we illustrated on the data structure queue that is simple enough for examples but quite rich for explaining behavioral concepts. We sketched out the comonadic approach based on the toposes and we showed how to construct coalgebraic operation in topos. The topos theory is a powerful tool for defining the behavior of program systems. Our future research will extend this approach with the Kleisli categories and monads and with the correct proving of the bisimilarity of the states. We also want to extend this approach to some paradigms of programming including object-oriented programming and logic programming.

Acknowledgment

This work was supported by VEGA Grant No.1/0175/08: Behavioral categorical models for complex program systems.

References

[1] Barr, M., and Wells, C. *Category Theory for Computing Science*. Prentice Hall International, 1990. ISBN 0-13-120486-6.

[2] Barr, M., and Wells, C. *Toposes, Triples and Theories*. Springer-Verlag, 2002.

[3] Ehrig, H., and Mahr, B. *Fundamentals of Algebraic Specification I.: Equations and Initial Semantics*. No. 6. EATCS, 1985. Monographs on Theoretical Computer Science.

[4] Ferrari, D. The improvement of program behavior. *Computer 9* (1976), 39–47.

[5] Hasuo, I. *Tracing Anonymity with Coalgebras*. PhD thesis, Radboud University Nijmegen, 2008.

[6] Jacobs, B. Introduction to coalgebra. *Towards Mathematics of States and Observations (draft)* (2005).

[7] Jacobs, B., and Rutten, J. A tutorial on (co)algebras and (co)induction. *Bulletin of the European Association for Theoretical Computer Science*, No. 62 (1997), 222–259.

[8] Johnstone, P. *Topos Theory*. Academic Press Inc., London, 1977.

[9] Mihályi, D. Behaviour of algebraic term sequences. In *Pietriková A., Slodičák V., Főző L. editors: 7th PhD Student Conference and Scientific and Technical Competition of Students of Faculty of Electrical Engineering and Informatics Technical University of Košice, Slovakia* (2007), FEI TU Košice, pp. 153–154. ISBN 978-80-8073-803-7.

[10] Novitzká, V., Mihályi, D., and Slodičák, V. Categorical models of logical systems in the mathematical theory of programming. In *MaCS'06 6th Joint Conference on Mathematics and Computer Science, Book of Abstracts* (2006), University of Pécs, Hungary, pp. 13–14.

[11] Novitzká, V., Mihályi, D., and Verbová, A. Coalgebras as models of systems behaviour. In *International Conference on Applied Electrical Engineering and Informatics, Greece, Athens* (2008), pp. 31–36.

[12] Novitzká, V. Logical reasoning about programming of mathematical machines. In *Acta Electrotechnica et Informatica* (March 2005), Košice, pp. 50–55.

[13] Phoa, W. *An introduction to fibrations, topos theory, the effective topos and modest sets.* The University of Edinburgh, 2006.

[14] Reichel, H. Behavioural equivalence - a unifying concept for initial and final specification methods. In *3rd Hungarian Computer Science Conference* (1981), no. 3, Akadémia kiadó, pp. 27–39.

[15] Schubert, C., and Dzierzon, C. Terminal coalgebras and tree-structures. In *18th Conference for Young Algebraists* (2003), University of Potsdam, Potsdam, Germany.

[16] Slodičák, V. *The Rôle of Toposes in the Informatics.* PhD thesis, Technical University of Košice, Slovakia, 2008. (in slovak).

[17] Slodičák, V. The toposes and their application in categorical and linear logic. In *6th PhD Student Conference and Scientific and Technical Competition of Students of Faculty of Electrical Engineering and Informatics Technical University of Košice* (2006), elfa s.r.o., pp. 121–122.

[18] Turi, D. *Category Theory Lecture Notes.* Laboratory for Foundations of Computer Science, University of Edinburgh, 2001.

[19] Wisbauer, R. *Algebras versus coalgebras.* University of Düsseldorf, Germany, 2007.

[20] Zhao, C., Kong, J., and Zhang, K. Program behavior discovery and verification: A graph grammar approach. *IEEE Transactions on Software Engineering 99*, PrePrints (2010).

Faculty of Electrical Engineering and Informatics, Technical University of Košice

E-mail address: viliam.slodicak@tuke.sk, valerie.novitzka@tuke.sk