JAM CM Journal of Applied Mathematics and Computational Mechanics 2016, 15(4), 145-156
www.amcm.pcz.pl p-ISSN 2299-9965
DOI: 10.17512/jamcm.2016.4.16 e-ISSN 2353-0588

SOME PROPERTIES OF COALGEBRAS AND THEIR RÔLE IN COMPUTER SCIENCE

William Steingartner[1], Davorka Radaković[2], František Valkošák[1], Pavol Macko[1]

[1] Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics
Technical University of Košice
Košice, Slovak Republic
[2] Department of Mathematics and Informatics, Faculty of Sciences, University of Novi Sad
Novi Sad, Serbia
william.steingartner@tuke.sk, davorkar@dmi.uns.ac.rs, frantisek.valkosak@student.tuke.sk
pavol.macko@gmail.com

Received: 31 July 2016; accepted: 10 November 2016

Abstract. This paper introduces basic theoretical knowledge of coalgebras in computer science. Coalgebras are, specifically in category theory, structures defined according to an endofunctor. For both algebra and coalgebra, a functor is a convenient and general way of defining a signature. We present practical usage of the coalgebras in an example. We observe a behavior of a simple Sequencer developed in SLGeometry framework. We model its behavior with the simple program written in Python, and we describe its behavior within coalgebra of endofunctor. The computation of the values stored in internal states is performed coinductively. Our approach can be used in the teaching process of formal methods for young software engineers.

Keywords: *coalgebra, functor, induction, internal state of program, observable behavior of program, SLGeometry software*

1. Introduction

Induction is a well-known mathematical proof method which is also used in computer science. The frequent usage of induction in computer science is with well-founded sets [1]. With these sets, we are able to define elements by constructors in finite steps. Induction is represented by notions like initial algebras and finality. Coinduction is dual notion to induction. It is also a proof method, which serves to gain some observations on structures and is used with non-well-founded sets or with infinite streams. Coalgebras are a very important part of theoretical computer science. Their main *rôle* is in describing the generated behavior of running programs [2, 3]. It is the behavior that can be observed from the outside of a machine - for instance on the screen. Typically, we model many problems by mathematical programs which seek to maximize or minimize some objective which is a function of the decisions [4, 5]. Coalgebra is an abstract notion of observable

behavior. It is a study of states and their operations and properties. The set of states is usually seen as a black box, to which one has limited access [6]. A relation between what is actually inside and what can be observed externally is the foundation of the theory of coalgebras [3].

The main goal of this article is to introduce the purpose of coalgebras in computer science and to introduce basic principles of coinductive computation.

The paper is organized as follows: Section 2 gives us basic notions of functor, algebra, coalgebra and category theory. The application of coinduction on Sequencer example is presented in Section 3. Experiment about Sequencer's behavior, checking of the results and description of Sequencer's behavior is introduced in Section 4. Finally, some conclusions are closing our article.

2. Basic concepts

In this section we briefly introduce some basic concepts that we use further.

Category theory is the algebra of functions. A *category* is an abstract structure: a collection of objects, together with a collection of arrows between them. Precisely, a category $\boldsymbol{C}$ is a mathematical structure that consists of a set of objects, e.g. $A, B, \ldots$ and a set of morphisms or arrows of the form $f : A \to B$ between them [7, 8].

For any given object A there is designated identity morphism $id_A : A \to A$. For two morphisms $f : A \to B$ and $g : B \to C$, there is designated composite morphism $g \circ f : A \to C$.

The objects of category can be arbitrary structures, therefore categories are useful in computer science, where we often use more complex structures not expressible by sets. Morphisms between categories are called *functors*, e.g. a functor $F : \boldsymbol{C} \to \boldsymbol{D}$ from a category $\boldsymbol{C}$ into a category $\boldsymbol{D}$ is considered as a structure-preserving mapping (identities, morphisms and composition) between categories [9, 10].

Let F be a functor. An *algebra* of the functor F is according to [11] a tuple (A, a) where:

- A is the carrier-set of the algebra;
- the function $a: F(A) \to A$ is the algebra structure.

Dually, an *F-coalgebra* or *F-system* is a pair (U, b) consisting of a set U and a function $\mathrm{b}: \mathrm{U} \to \mathrm{F}(\mathrm{U})$ [12, 13]. The set U is called the carrier of the system, also called the set of states; the function b is called the *F-transition structure* (or dynamics) of the system. When no explicit reference to the functor (i.e., the type of the system) is needed, we shall simply speak of *system* and *transition structure*. Moreover, when no explicit reference to the transition structure is needed, we shall often use U instead of (U, b).

Coalgebras are dual notion to algebras. We show a difference between them in an example. What is the main difference between an algebra $a: F(A) \to A$

and a coalgebra $b: U \to F(U)$? Essentially, the difference is between construction and observation. An algebra consists of a carrier-set A with a function $a: F(A) \to A$ going into this carrier A. It tells us how to construct elements in A. Coalgebra consists of a state space U with a function $b: U \to F(U)$ going out of U, in an opposite direction. In this case we do not know how to form elements in U, but we only have operations acting on U, which may give us some information about U [9].

According to [13, 14], the main differences between the theory of universal algebra and universal coalgebra is shown in Table 1.

Table 1

Comparision of universal algebra and coalgebra

Universal algebra	coinduction
$(\sum -)$algebra	coalgebra = system
algebra homomorphism	system homomorphism
substitutive relation	bisimulation relation
congruence	bisimulation equivalence
subalgebra	subsystem
initial algebra (is minimal, plus induction definition principle)	initial system (often trivial)
final algebra (often trivial)	final system (is simple, plus: coinduction definition principle)

3. Coalgebras and SLGeometry

In this section we introduce SLGeometry framework with its UI controls. In [15] we demonstrated the use of triggers and their propagation from one component to another as children's mathematical game to calculate arithmetic sum. We present how this game can be described in a formal way, as well as how it could be used for teaching coalgebras at the faculty level.

3.1. SLGeometry

SLGeometry is an extensible dynamic geometry framework, which is now implemented as a Universal Windows Application on the .NET Framework [15-18]. It consists of an expression parser and an extensible functional language (*Engine*) at its core, and a graphical surface (*GeoCanvas*). The main purpose of the *Engine* is to maintain a set of expressions, stored in named variables, which represent the elements of geometric drawings.

Geometric shapes and visual controls are displayed by *GeoCanvas* as a response to user interaction. Geometric drawings are designated by expressions, built from

atomic values, lists and functions applied to parameters. A dynamic of a system is evident with each change of a particular variable, then a recursive recalculation of all dependent variables is triggered. Since the variables are bound to graphical objects and user controls on the screen, the user can change a variable by interacting with the bound object. After that, all dependent objects are recalculated and their corresponding graphical objects on the *GeoCanvas* are updated.

The main aim of the SLGeometry framework is to improve the development of teaching materials and games using it as a foundation for experiments. The motivation for developing custom UI controls was found on GeoGebraTube (now on [19] with more than 450.000 of free and interactive materials) where many teaching materials and games could be found. The representation of each of these controls is given with a single variable and all controls have properties which determine the control's appearance and behavior.

In [15] we proposed UI controls with a sequential behavior Sequencer and Scorekeeper to demonstrate how to calculate the arithmetic sum of numbers from 1 to 9. A Sequencer is a counter which cycles through an interval of integer values and it behaves like modulo. The interval in the Sequencer can be assigned. An UI control Scorekeeper keeps added points when it is triggered. The user clicks on a PushButton which then triggers the Sequencer and it's Done property triggers the Scorekeeper. To achieve this flow, the given controllers has to be connected to each other by the user. If the Sequencer is denoted with variable `s`, PushButton with `p` and Scorekeeper with `k`, the connection is given in Table 2.

Table 2

Daisy chaining via triggers

```
s.Trigger = p.Pressed
k.Points = s.Value
k.Trigger = s.Done
```

3.2. Formal description of Sequencer

We use the given example of controllers to demonstrate the modulo as an example of sequential behavior that is attractive for the students. It propagates coinductive computation and coalgebras in an interesting way. In this case, we use the interval of naturals from 0 to 8, so Sequencer properties Min and Max are changed to 0 and 8, respectively. This represents the congruence class of the modulo 9, denoted as A for the presentation of coinduction with this example. Whenever the PushButton is pushed, the Sequencer increments the value by 1.

We use derived function $h: \mathbb{N} \times \mathbb{N} \to A$, which sends its input value to the congruence class of the modulo a:

$$h(n, a) = n \bmod a.$$

In our case the value of a is 9, i.e. $a = 9$:

$$h(n, 9) = n \bmod 9.$$

The value of the Scorekeeper is computed as follows:

$$h(i,a) + \sum(pred(i)), \tag{1}$$

where i stands for index and a for quotient value for modulo calculation.

We define a state for any a as follows:

$$s = \begin{cases} (i, h(i,a) + \sum(pred(i))), & if\ i > 0; \\ (0,0), & otherwise. \end{cases} \tag{2}$$

A state is defined as a tuple of two values:

- value i is a number of how many times the PushButton has been pushed;
- $\sum(i')$ is a sum of all values given by Sequencer from 0 to i' and kept by the Scorekeeper.

The states we enclose into the state space X. We define destructor operations over the state space as follows:

$$value : X \to \mathbb{N}$$

$$index : X \to \mathbb{N}$$

$$next : X \to X$$

where $\mathbb{N}$ is a set of naturals. An operation $value$ returns the actual sum stored in a state; it is an operation that represents the behavior of the Sequencer. The operation $index$ is triggered by the PushButton and it autoincrements the index of state. The last operation $next$ generates a new state. The new state appears after pushing the button and the value of the index is incremented by 1. This could be expressed in a formal way for a state $s = (i, h(i,a) + \sum(pred(i)))$ as follows:

$$s' = next(s) = (succ(i), h(succ(i), a) + \sum(i))$$

$$index(s') = succ(i)$$

$$value(s') = h(succ(i), a) + \sum(i)$$

We enclose functions $value$ and $next$ into the coalgebraic specification as follows:

Operations:

$$value : X \to \mathbb{N}$$

$$index : X \to \mathbb{N}$$

$$next : X \to X$$

Assertions:

$$value\left(next\left(next(s)\right)\right) = value\left(next(s)\right) + h(index\left(next\left(next(s)\right)\right), a)$$

Creation:

$$value(new) = 0$$

$$index(new) = 0$$

$$value\left(next(new)\right) = 1$$

$$index\left(next(new)\right) = 1$$

We define the coalgebra structure as follows:

$$\langle value, index, next\rangle \colon \mathrm{X} \to \mathbb{N} \times \mathbb{N} \times \mathrm{X}.$$

The appropriate endofunctor for the coalgebra is

$$F(\mathrm{X}) = \mathbb{N} \times \mathbb{N} \times \mathrm{X}.$$

By observing the output values of the Sequencer we can formulate the following proposition.

Proposition 1: Let s be a state with $index(s) = i$ and $value(s) = v$ (according to (2)) and let $a \in \mathbb{N}$ be a value for modulo calculation.

If the following congruence

$$i \equiv a - 1 \ (mod\ a),$$

holds, then in the state $s' = next(s)$ with $i' = index(s')$ and $v' = value(s')$ the corresponding values are the same:

$$value(s) = value(s'),$$
$$\text{i.e. } v = v'.$$

Proof: The proof is done by induction.

(1) In the case $a = 1$, for any state s let index be some value i, $index(s) = i$. It holds, that $value(s) = h(i, 1) + \Sigma\left(pred(i)\right) = 0$. Then for the state $s' = next(s)$ holds that $index(s') = i + 1$ and $value(s') = 0$ (according to (2)).

Thus, the property for $a = 1$ holds trivially.

(2) Now we consider, that for some $a = k \in \mathbb{N}$ an induction hypothesis

$$value(s) = value(s')$$

holds and we prove the property for $a = k + 1$.

We present for $a = k$ what holds. Let $v_i = value(s)$ and $v_{i+1} = value(s')$. According to the formal definition (2) at the state with the index $i + 1$ it holds that

$$v_{i+1} = h(i + 1, k) + v_i.$$

Since,

$$i \equiv k - 1 \ (mod\ k) \text{ (by definition)}$$

and

$$1 \equiv 1 \ (mod\ k) \text{ (reflexivity of congruence by modulo } k\text{)},$$

by addition of congruence then we have

$$\begin{gathered} i + 1 \equiv k(mod\ \mathrm{k}) \Longrightarrow \\ i + 1 \equiv 0 \ (mod\ k) \Longrightarrow \\ h(i + 1, a) = 0, \\ \text{i.e.} \\ v_{i+1} = v_i. \end{gathered}$$

The proof can also follow the following idea: for any $i \in \mathbb{N}$, the sequential state values are:

$$\begin{aligned} v_i &= h(i, k) + v_{i-1}, \\ v_{i+1} &= h(i + 1, k) + v_i. \end{aligned}$$

We denote a modulo (or remainder) operator simply $a\ mod\ b$, where b be a quotient. Let i be a value

$$i = pk + (k - 1), \tag{3}$$

which is alternatively

$$i\ mod\ k = k - 1$$

and $k, p \in N$.

Adding value 1 to both sides of (3) we get

$$\begin{aligned} i + 1 &= pk + (k - 1) + 1 \\ &= pk + k \\ &= k(p + 1) \end{aligned}$$

which follows to

$$(i + 1)\ mod\ k = 0.$$

Then we can express state values as follows:

$$v_i = h(i,k) + v_{i-1} = k - 1 + v_{i-1}, \qquad (4)$$
$$v_{i+1} = h(i+1,k) + v_i = 0 + v_i = k - 1 + v_{i-1}.$$

From induction hypothesis it follows that state values v_i and v_{i+1} are equal,

$$v_i = v_{i+1},$$

if $i \; mod \; k = k - 1$ for given $k \in \mathbb{N}$.

Now we consider the following congruence for $a = k + 1$:

$$i \equiv k + 1 \; (mod \; (k+2)).$$

Then we can express state values according (4) for $a = k + 1$ as follows:

$$v_i = h(i,k+2) + v_{i-1} = k + 1 + v_{i-1},$$
$$v_{i+1} = h(i+1,k+2) + v_i = 0 + v_i = k + 1 + v_{i-1},$$

which means that state values v_i and v_{i+1} are equal,

$$v_i = v_{i+1},$$

if $i \; mod \; (k+1) = k$ for given $k \in \mathbb{N}$.

■

Hence, we formulate the following corollaries.

Corollary 1: First occurrence of value v and v' such that $v = v'$, defined in Proposition 1 is at index with value $a - 1$.

Corollary 2: The given tuples appear after each *a* steps.

4. Experiment and results

In this section we present an example of the Sequencer's behavior. We follow the presented example in [15]: we model it in a formal way and then we show its behavior by a simple program written in the Python programming language. Finally, we compare the results when the quotient is changed.

At the beginning, the Scorekeeper is in the initial state s_0 and its sum value is 0, i.e. $s_0 = (0,0)$. After the PushButton is pushed, the next state is generated and its value is previous index value incremented by 1. So after the PushButton is pushed ten times, the sequence of ten new states appear according to the following formula:

$$s_{i+1} = next \; (s_i), \text{ where } 0 \leq i \leq 9 \text{ in our example.}$$

The list of states then looks like:

$$s_0 = (0,0) \qquad s_6 = (6,21)$$
$$s_1 = (1,1) \qquad s_7 = (7,28)$$
$$s_2 = (2,3) \qquad s_8 = (8,36)$$
$$s_3 = (3,6) \qquad s_9 = (9,36)$$
$$s_4 = (4,10) \qquad s_{10} = (10,37)$$
$$s_5 = (5,15)$$

4.1. Simulation of Sequencer's behavior

To check the Proposal in practice, we also demonstrated the Sequencer's behavior by a simple program in the Python programming language. The source code is listed in Table 3.

Table 3

Python listing of Sequencer's behavior

```
1.  import sys
2.
3.  def h(n, a):
4.      return (n % a)
5.
6.  def seq(n, a):
7.      if (n == 0):
8.          return (0)
9.      else:
10.         return (seq(n - 1, a) + h(n, a))
11.
12. f = 0
13. a = 9
14.
15. for i in range(30):
16.     print("index={}, value={}".format(i, f))
17.     f = seq(i, a)
```

We conducted an experiment for different values of a (see line 13 in Table 3). Tables 4 and 5 show the concrete computed values for $a = 5$ and $a = 9$ (as in SLGeometry), for simplicity in the first 30 states. In both cases we can see the values as they are generated by the Sequencer. The property in Proposition 1, which we proved, can also be observed on the output values.

Table 4

Values for a = 5

index=0, value=0	index=15, value=30
index=1, value=1	index=16, value=31
index=2, value=3	index=17, value=33
index=3, value=6	index=18, value=36
index=4, value=10	index=19, value=40
index=5, value=10	index=20, value=40
index=6, value=11	index=21, value=41
index=7, value=13	index=22, value=43
index=8, value=16	index=23, value=46
index=9, value=20	index=24, value=50
index=10, value=20	index=25, value=50
index=11, value=21	index=26, value=51
index=12, value=23	index=27, value=53
index=13, value=26	index=28, value=56
index=14, value=30	index=29, value=60

Table 5

Values for a = 9

index=0, value=0	index=15, value=57
index=1, value=1	index=16, value=64
index=2, value=3	index=17, value=72
index=3, value=6	index=18, value=72
index=4, value=10	index=19, value=73
index=5, value=15	index=20, value=75
index=6, value=21	index=21, value=78
index=7, value=28	index=22, value=82
index=8, value=36	index=23, value=87
index=9, value=36	index=24, value=93
index=10, value=37	index=25, value=100
index=11, value=39	index=26, value=108
index=12, value=42	index=27, value=108
index=13, value=46	index=28, value=109
index=14, value=51	index=29, value=111

Comparing the results of the Sequencer in Table 4 (a = 5) and in Table 5 (a = 9) we can see the following:

– state values in which $i \bmod a = a - 1$ have the same values for index i and index $i+1$;
– the first occurrence of values v and v' such that $v = v'$, defined in Proposition 1 is at index with value $a - 1$ in accordance with the Corollary 1;
– and these given tuples appear after each a steps in accordance with the Corollary 2.

5. Conclusion

The main goal of this article was to bring a closer view on coalgebras in computer science. We introduced some example and practical examples of coalgebras and coinductive computation of state values. The example used with SLGeometry

represents a very interesting way of using coalgebras and coinduction. We want to extend our approach with the coinduction and bisimulation method when observing the output states of a running system.

Acknowledgement

This work was supported by Grant No. FEI-2015-18: Coalgebraic models of component systems.

&

This work is the impact of the projects implementation: Center of Information and Communication Technologies for Knowledge Systems (ITMS project code: 26220120020 and Development of the Center of Information and Communication Technologies for Knowledge Systems (ITMS project code: 26220120030) supported by the Research Development Operational Program funded by the ERDF.

&

This work is a result of an international cooperation under the CEEPUS network No. CIII-HU-0019-11-1516.

References

[1] Aczel P., Non-well-founded sets, Center for the Study of Language and Information, Stanford 1941.

[2] Adámek J., Milius S., Moss L.S., Initial algebras and terminal coalgebras: a survey, 2010.

[3] Jacobs B., Rutten J., An introduction to (co)algebras and (co)induction, [in:] D. Sangiorgi, J. Rutten (eds.), Advanced topics in bisimulation and coinduction, 2011 (This chapter is a modest update of the 1997 BEATCS old tutorial paper).

[4] Novitzká V., Slodičák V., On applying stochastic problems in higher-order theories, Acta Electrotechnica et Informatica 2007, 7, 3.

[5] Ristić S., Aleksić S., Čeliković M., Luković I., Generic and standard database constraint metamodels, Comput. Sci. Inf. Syst. 2014, 11(2), 679-696.

[6] Jacobs B., Introduction to Coalgebra. Towards Mathematics of States and Observations, Version 2.0, 2012.

[7] Walters R.F.C., Categories and Computer Science, Cambridge University Press, New York 1992.

[8] Brandenburg M., Einführung in die Kategorientheorie. Mit ausführlichen Erklärungen und zahlreichen Beispielen, Springer-Verlag, Berlin-Heidelberg, Springer Spektrum, 2016.

[9] Barr, M., Wells, C., Category Theory for Computing Science, Prentice Hall International, 1990.

[10] Blute R., Scott P., Category theory for linear logicians, [in:] T. Erhard, J.-Y. Girard, P. Ruet (eds.), Linear Logic in Computer Science, London Mathematical Society Lecture Note Series, Cambridge Univ. Press, 2004.

[11] Geuvers H., Representing Streams in Second Order Logic (Coinduction and Coalgebra in Second Order Logic), Seminar Representing Streams, Lorentz Centre, Leiden Dec 14 2012, ISBN 3-540-66463-7.

[12] Vene V., Categorical Programming with Inductive and Coinductive Types, Tartu University Press, Tartu 2000.

[13] Rutten J.J.M.M., Fundamental study - universal coalgebra: a theory of systems, Theoretical Computer Science 2000, 249, 3-80.

[14] Jacobs B., Rutten J., Tutorial on (Co)Algebras and (Co)Induction, EATCS 1999, 222-259.

[15] Radaković D., Herceg Đ., A platform for development of mathematical games on silverlight, Acta Didactica Napocensia 2013, 6(1), 77-90.

[16] Radaković D., Herceg Dj, The Extensibility of an Interpreted Language Using Plugin Libraries, AIP Proceedings, Numerical Analysis and Applied Mathematics ICNAAM 2011: International Conference on Numerical Analysis and Applied Mathematics 2011, 1389, 837-840.

[17] Herceg Đ., Radaković D., Herceg D., Generalizing the Extensibility of a Dynamic Geometry-Software, AIP Proceedings, Numerical Analysis and Applied Mathematics ICNAAM 2012: International Conference of Numerical Analysis and Applied Mathematics 2012, 1479, 482-485.

[18] Radaković D., Herceg Dj, The Use of WPF for Development of Interactive Geometry Software, Acta Universitatis Matthiae Belii, Series Mathematics 2010, 16, 65-79.

[19] GeoGebra: Featured Materials, https://www.geogebra.org/materials/