Some useful aspects about coalgebræ and coalgebraic logic in computer science

Ján Perháč, Daniel Mihályi, William Steingartner, Valerie Novitzká
jan.perhac@tuke.sk, daniel.mihalyi@tuke.sk,
william.steingartner@tuke.sk, valerie.novitzka@tuke.sk

Technical University of Košice
Faculty of Electrical Engineering and Informatics
Košice, Slovak Republic

Abstract

Nowadays, it is difficult to describe any algebraic dynamic structures that usually includes the notion of state. A suitable structure for describing these systems can be a coalgebra, properties of which are dual to algebraic by their modeling in terms of category theory. This paper presents our approach for a formal description of a program system behavior based on category theory, coalgebræ and coalgebraic logic. At the beginning, we briefly present basic notions from mentioned formal methods, and then we present motivation example from the area of the operating system GNU/Linux by its system calls.

1 Introduction

Coalgebræ are useful categorical structures with many applications in computer science. In our approach, they are used for modeling state oriented behavior of program systems.

Many logical systems, which are used in computer science, are derivations of modal logics. In our paper we present one of them, which is based on category theory and coalgebræ, named coalgebraic logic. Its first definition came from Moss [Mos97]. It is necessary for definition of this logical system to define category of classes and polynomial endofunctor over such a category.

Coalgebraic logic, with its expressive power, has a wide use in computer science through its formulæ, for example, specifying non-well founded abstract data structures [Kur01], infinite data structures [Kur00], or observation of program systems behavior [Mih14].

Our recent works regarding coalgebræ [Mih12], [Mih14], [Per15] were oriented to modeling of intrusion detection system behavior as coalgebra for an appropriate endofunctor over category of infinite packet stream. We present in this work several fruitful aspects about coalgebraic logic and its role in a formal description of program behavior.

2 Basic notions

Definition of the coalgebraic logic is based on category theory and coalgebræ [Mos97], therefore we briefly introduce basic notions from mentioned mathematical structures in this section. First, we bring definition of category theory, and second, we define basic notions from coalgebræ , which are based on this theory.

In: E. Vatai (ed.): Proceedings of the 11th Joint Conference on Mathematics and Computer Science, Eger, Hungary, 20th – 22nd of May, 2016, published at http://ceur-ws.org

2.1 Category theory

Category theory was founded in 1945 by Eilenberg's and MacLane's work as a part of their research in the field of algebraic topology [Mac45], where they formulated a notion of categories, functors and natural transformations of functors, as a formalism for describing mathematical structures. Nowadays [Mih10], category theory is a universal abstract mathematical frame used for description of various structures as mathematical, algebraic, or abstract data structures used in computer science.

According to [Bar98], [Jac97], a category $\mathbf{C}$ is a mathematical structure which consists of:

- a class of objects $C_o \in \mathbf{C}$, where $C_o = \{X, Y, Z \dots\}$ and
- a class of morphisms between objects $C_m \in \mathbf{C}$, where $C_m = \{f, g, h \dots\}$.

Every category $\mathbf{C}$ possesses the following well defined properties:

- If there is a morphism $f \in C_m$ between two objects $X, Y \in C_o$ from the object X to the object Y it is denoted as $f : X \to Y$.
- Lets have a morphism $f \in C_m$ defined as $f : X \to Y$. Then object $X \in C_o$ is a domain of a morphism and object $Y \in C_o$ is its codomain.
- For every object $X \in C_o$ there is an identity morphism $id_X \in C_m$ defined as: $id_X : X \to X$, where domain and codomain of such a morphism is X.
- The most important property of every category $\mathbf{C}$ is that morphisms are composable i.e. lets have $X, Y, Z \in C_o$ and two morphisms $f, g \in C_m$ define as follows:

  $f : X \to Y$ and $g : Y \to Z$, then there is a morphism $g \circ f : X \to Z$.

Typical example of a category is the category of classes **SET**, where objects are represented as sets and morphisms as functions between them.

2.2 Functors

Many mathematical structures can be modeled as categories [Ste14]. Since categories are also mathematical structures [Kur01], one should consider a category of categories, where objects are categories and morphisms of such a category are a structure preserving maps between categories, called functors.

Let $\mathbf{C}$ and $\mathbf{D}$ be categories and $F : \mathbf{C} \to \mathbf{D}$ be a functor. Then F is a morphism from the source category $\mathbf{C}$ to the destination category $\mathbf{D}$ [Kur00]. A functor F assigns to every object $X \in \mathbf{C}$ an object $F(X) \in \mathbf{D}$. Also a functor F assigns to every morphism $f : X \to Y \in \mathbf{C}$ a morphism $F(f) : F(X) \to F(Y) \in \mathbf{D}$ satisfying following conditions:

- for every object $X \in \mathbf{C}$ and its identity object morphism $id_X : X \to X$ there is

$$F(id_X) = id_{F(X)};$$

- for every two morphisms $f, g \in \mathbf{C}$ a functor F preserves composition properties $(g \circ f)$ such as:

$$F(g \circ f) = F(g) \circ F(f).$$

This kind of functors is called simple [Bar98]. Another examples of simple functors are:

- identity functor $ID_{\mathbf{C}} : \mathbf{C} \to \mathbf{C}$;
- endofunctor $E : \mathbf{C} \to \mathbf{C}$, where source and destination category are the same.

In our approach, we use a special kind of endofunctor called polynomial endofunctor [Per15], [Mih12], [Slo11]. A polynomial endofunctor over category $\mathbf{C}$ is an endofunctor formed by polynomial operations shown in Backus-Naur form (BNF) production rule as follows:

$$F(X) ::= X | X \times Y | X + Y | X^Y. \tag{1}$$

2.3 Coalgebra for a polynomial endofunctor

Coalgebræ are useful categorical structures in computer science [Mac11], [Sma11]. In our approach we use them for modeling state oriented behavior of program systems.

Lets have a category $\mathbf{C}$ and polynomial endofunctor F [Kur01], [Mos97], [Kur00], then a coalgebra for a polynomial endofunctor F (sometimes also called F-coalgebra) is formally an ordered pair:

$$(X, \zeta), \tag{2}$$

where

- X is a coalgebra's state space of objects of category $\mathbf{C}$ and
- ζ is a morphism of category $\mathbf{C}$ such as $\zeta : X \to F(X)$.

The morphism ζ is also called a structural function of a coalgebra.

3 Coalgebraic logic

A plethora of current system behavior researches had shown the importance of choosing a suitable logical language. Those formulæ are used to logical reasoning over the states of the dynamic system. The states are captured by the coalgebra of an appropriate polynomial endofunctor.

Based on our previous work, in this chapter we present logical system based on multimodal language. It is suitable for description of the behavior of infinite data structures. This logic is called the coalgebraic logic introduced by Moss in his work [Mos97]. For its definition, it is necessary to define the category of classes **SET** and polynomial endofunctor over it as $F : \mathbf{SET} \to \mathbf{SET}$.

Note 3.1: Since we won't be manipulating only with formulæ and sets of formulæ [Bil13], but also with elements of $F_\omega L_\omega$, it is beneficial to use the convention of notations from the table 1. To be more transparent, we will use notation F_ω for polynomial endofunctor instead of F.

Table 1: Conventions of notations in coalgebraic logic

Set	Elements
L_ω	$\varphi, \psi, \ldots$
$F_\omega L_\omega$	$\Phi, \Psi, \ldots$

3.1 Syntax of coalgebraic logic

Kurz in his work [Kur01] says that the main reason for introduction of the coalgebraic logic is to find a modal language L_ω for every coalgebra (X, ξ) and to specify its satisfiability relation as follows

$$\vDash_{F\omega} \subset X \times L_\omega, \tag{3}$$

where

- if $\Phi \subset L_\omega$, then $\wedge\Phi \in L_\omega$;
- if $\Phi \subset L_\omega$, then $\neg\varphi \in L_\omega$;
- if $\varphi \in F_\omega L_\omega$, then $\varphi \in L_\omega$.

Based on the definition above, it is clear that the $\wedge\{\} \in L_\omega$, the clause $\bigwedge\{\}$ denotes *true* and L_ω is a proper class [Kur00]. We abbreviate $\bigwedge\{\}$ as $\top$. The last clause uses fact that the F_ω is a functor over **SET**.

The modal language L_ω of coalgebraic formulæ can be expressed by following production rule in Backus-Naur form:

$$\varphi ::= \bot \mid \top \mid \neg\varphi \mid \bigwedge\Phi \mid \nabla\Phi, \tag{4}$$

where $\varphi \in L_\omega$ and $\Phi \in F_\omega L_\omega$.

From the production rule (4), it is clear that formulæ of coalgebraic logic are ordered pairs or their infinite conjunctions [Mih10]. While the modality $\nabla\Phi$ means shorthand notation

$$\nabla\Phi \equiv \Box\bigvee\Phi \wedge \bigwedge\Diamond\Phi, \tag{5}$$

where $\bigwedge, \bigvee$ are infinite forms of conjunction and disjunction. The following equivalences are valid:

$$\begin{array}{lcl} \Diamond\varphi & \equiv & \nabla\{\varphi, \top\} \\ \Box\varphi & \equiv & \nabla\Phi \vee \nabla\{\varphi\} \end{array} \tag{6}$$

Example 3.1.1 Let $F_\omega X = A \times X$ and $a_i \in A$, $i \in \mathbb{N}$.
Then the formulæ

$$(\top), (a_0, \top), (a_0(a_1, \top), \ldots, \bigwedge\{(a_0, a_1, \ldots, a_n, \top) \mid n \in \mathbb{N}\} \tag{7}$$

are formulæ of coalgebraic logic.

3.2 Semantics of coalgebraic logic

Let (X, ξ) be a coalgebra and $x \in X$. Then according to [Kur00] the semantics of coalgebraic logic can be expressed by the following satisfiability relation

$$\vDash_{F\omega} \subset X \times L_\omega, \tag{8}$$

as follows:

$$\begin{array}{lll} x \vDash_{F\omega} \varphi, \forall\varphi \in \Phi, \Phi \subset L_\omega & \textit{then} & x \vDash_{F\omega} \wedge\varphi \\ x \nvDash_{F\omega} \varphi & \textit{then} & x \vDash_{F\omega} \neg\varphi \\ \exists w \in F_\omega(\vDash_{F\omega}) \textit{ so, } F_\omega\pi_1(w) = f(x), \quad F_\omega\pi_2(w) = \varphi & \textit{then} & x \vDash_{F\omega} \varphi \end{array} \tag{9}$$

where π_1, π_2 are projections of product $X \times L_\omega$.

4 Motivation example

We present traceability of intrusion system calls of a running functional program under the OS GNU/Linux as a demonstration example [Mra13]. We understand system calls as infinite stream, therefore it is suitable to model them as coalgebra for a polynomial endofunctor. We also specify system calls as formulæ of coalgebraic logic.

4.1 Many-typed coalgebraic signature of system calls

First, we define many-typed coalgebraic signature of system calls. Formally it is an ordered pair

$$\Sigma = (\mathcal{T}, \mathcal{F}), \tag{10}$$

where

- $\mathcal{T}$ is a class of types names and
- $\mathcal{F}$ is a class of names of operations over types.

BEGIN Signature
Σ_s
Begin types
$\mathcal{T} = \{action, char, hex, int, nat\}$
end
Begin opns
$\mathcal{F} = \{alert, access, overflow :\rightarrow actions,$
$name :\rightarrow char,$
$arguments :\rightarrow char,$
$rt\_sigproc : char \times nat \times char \times nat \rightarrow int,$
$rt\_sigprocmask :\rightarrow int,$
$clock\_gettime :\rightarrow int,$
$clone : char \times char \times char \rightarrow nat,$
$timer\_set : int \times int \times char \times char \times char \rightarrow int,$
$mmap : char \times nat \times char \times char \times nat \times int \rightarrow hex,$
$munmap : hex \times nat,$
$wr\_rd : nat \times char \times nat \rightarrow int,$
$open : char \times char \times nat \rightarrow int,$
$setuid : int \rightarrow int,$
$returnValue :\rightarrow int,$
$nReturnValue :\rightarrow nat,$
$hReturnValue :\rightarrow hex$ }
end
END

The names of basic types from the class $\mathcal{T}$ in signature Σ_s define types of observed system calls [Mra13]. Operational specifications, from the class $\mathcal{F}$ in signature Σ_s, define the structure (syntax) of individual system calls. The description of individual system calls is specified in the table 2.

Table 2: System calls specification

Name	Description
name	name of the system call
arguments	the system call arguments
rt_sigproc	structure of the system call for working with memory
rt_sigprocmask	examine and change blocked signals
clock_gettime	get the current time
clone	structure of the system call to create a new process
timer_set	structure of the system call for the time of process
mmap	structure of the system call to initialize memory
munmap	structure of the system call to release memory
wr_rd	structure of the system call for working with the files - read/write
open	structure of the system call related to open file operation
returnValue	system call return value (integer)
nReturnValue	system call return value (non-negative integer)
hReturnValue	system call return value (hexadecimal number)

4.2 Category of system calls

In the second step, we construct a category of system calls $\mathcal{S}calls$ [Mra13], where

- *objects* are system calls $s_1, s_2, \ldots$;
- *morphisms* $next : s_i \rightarrow s_{i+1}$ are transition into the next system call of a given stream.

4.3 Polynomial endofunctor over category of system calls

We specify a polynomial endofunctor over a category of system calls $\mathcal{S}calls$ [Mra13] as follows

$$E : Scalls \rightarrow Scalls. \tag{11}$$

We define actions of the endofunctor as follows

$$E(s) : O \times s, \tag{12}$$

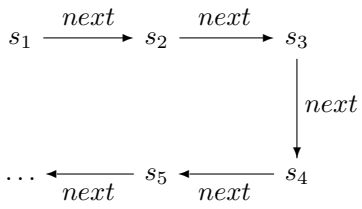

Figure 1: Category of infinite stream of system calls $\mathcal{S}calls$

$$E(next(s)) : O \times next(s), \tag{13}$$

where O denotes observable values of a given system call.

4.4 System calls as coalgebra

Finally we model a stream of system calls as a coalgebra for a polynomial endofunctor. For demonstration we present two intrusion system call sequences.

$\mathbb{A}$ denotes **Overloading of HW resources** in this case by fork processes [Mra13]. Symptoms of this disruption are placed in a generated stream of system calls, where the system calls are repeated sequentially:

```
name = rt_sigprocmask ; nReturnValue = 0
name = clock_gettime ; nReturnValue = 0
name = clock_gettime ; nReturnValue = 0
name = clock_gettime ; nReturnValue = 0
name = rt_sigprocmask ; nReturnValue = 0 ....
```

$\mathbb{B}$ denotes **Permission denied** i.e. security violation of the stored data. Symptoms of this disruption are placed in the following generated stream of system calls.

```
name = open, nReturnValue = -1  access to file
name = write; wr_rd = nat, char, 4 ;
nReturnValue = 63 unauthorized writing to file
name = read; wr_rd = nat, char, 2 ;
nReturnValue = 68 unauthorized reading to file
```

Coalgebra (system) observes a stream of system calls and if it detects symptoms of disruption, then it responds by making one of the following preferred (re)actions (actions from the Σ_s) [Mra13], such as:

- $\mathbb{A}lert$ - generates attention on the screen;
- $\mathbb{W}rong$ - unauthorized access, which saves output into appropriate log file and
- $\mathbb{O}verflow$ - indication of the resource misuse, i.e. termination of whole implementation (EXIT_FAILURE).

In the last step [Mra13], we define coalgebra with the detection of intrusion system activities as follows

$$(s_s, \langle hd, tl, N\_signal \rangle), \tag{14}$$

which is characterized by the following operations

- $hd : s_s \to s$ – immediate observation of the given system call in actual state;
- $tl : s_s \to s_s$ – state modification and

- generation of appropriate action ($N\_signal : s_s \to s^{actions}$) in the from

$$\langle hd, tl, N\_signal \rangle : s_s \to s \times s_s \times s^{actions^N} \tag{15}$$

where $s^{actions^N}$ expresses the generation of the appropriate reaction based on the given intrusion N e.g. $\mathbb{A}$ or $\mathbb{B}$.

4.5 Specification of the stream of system calls by coalgebraic logic

The formulæ of coalgebraic logic are used for logical reasoning over states of a dynamic system that is captured by the coalgebra for a polynomial endofunctor [Mih14]. Now we specify infinite stream of system calls as formulæ of the modal language defined in the section 3.

The application of coalgebraic specification creates an infinite sequence of coalgebraic formulæ

$$\begin{aligned} &(\top) \\ &(s_1, \top) \\ &(s_1, (s_2, \top)) \\ &(s_1, (s_2, (s_3, \top))) \\ &\dots \\ &\bigwedge\{(s_1, (s_2, (s_3, (\dots, (\dots, \top)\dots))))\} \end{aligned} \tag{16}$$

where the first line $(\top)$ denotes an empty sequence, which is the initial state of the system. The second line rises after the first application of coalgebraic specification. It specifies the initial (*starting*) system call. The next lines describe iterative application of coalgebraic specification up to a possible infinite sequence.

4.6 Example of detected intrusion system call

Behavior of the system can be modeled step by step by the following sequence

$$\begin{aligned} & (s_{start}, s\dots) \mapsto \\ \mapsto \quad & ((s_{rt\_sigproc}, s_{clock\_gettime}, s_{clock\_gettime}, s_{clock\_gettime}, s_{rt\_sigproc}), \epsilon) \mapsto^{10} \\ \mapsto \quad & ((s_{rt\_sigproc}, s_{clock\_gettime}, s_{clock\_gettime}, s_{clock\_gettime}, s_{rt\_sigproc}), \mathbb{A} \mapsto \mathbb{A}lert) \mapsto \\ \mapsto \quad & \dots \end{aligned}$$

In this example, stream starts with the system call $s\_start\dots$, which represents initialization system calls of a real system tool for system calls tracing (*Strace*) [Mra13]. Next system calls *s_timer_set*, *s_clone* are specific types of system calls. The coalgebra observes the stream and detects intrusion of the type $\mathbb{A}$ and responds by the action $\mathbb{A}lert$). The appropriate sequence of a real system calls is:

```
...
Initialization of Strace system calls.
...
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
clock_gettime(CLOCK_THREAD_CPUTIME_ID, {0, 4854270}) = 0
clock_gettime(CLOCK_MONOTONIC, {2935465, 569821866}) = 0
clock_gettime(CLOCK_PROCESS_CPUTIME_ID, {0, 4817026}) = 0
rt_sigprocmask(SIG_BLOCK, [INT], [], 8) = 0
...
```

Similarly, we can model the $\mathbb{B}$ or any known intrusion system call.

5 Conclusion

In this contribution we briefly present basic notions from category theory, coalgebræ and coalgebraic logic. We demonstrate our approach of a formal description of program systems behavior on the example from operating system's GNU/Linux detection and handling of intrusion system calls.

We see a possibility in creating verifiable model, using mathematical formal methods to develop intrusion detection systems by observing their behavior. Based on this, it is possible to eliminate undesirable behavior

and, with a combination of host and network intrusion detection systems, to create a complex security for program systems.

In the future, we would like to extend our approach by joining the network intrusion detection systems with a host-based one. A combination of both types of IDSs will allow us to secure computer systems even more. Next step in our work will be extending IDS by applying the resource oriented Belief-Desire-Intention logical system, that will allow automated IDSs reactions, instead of a system administrator intervention. Another idea for our future work is an application of a self-healing code procedures, which would allow a detection and handling of even new and unknown intrusions.

Acknowledgment

This paper was supported by KEGA project ViLMA: Virtual Laboratory for Malware Analysis (079TUKE-04/2017).

This work is a result of international cooperation under the CEEPUS network No.CIII-HU-0019-12-1617.

References

[Bar98] M. Barr, and C. Wells. *Category theory for computing science.* Prentice Hall International (UK) Ltd., 66 Wood Lane End, Hertfordshire, UK, 1998.

[Bil13] M. Bilková, A. Palmigiano, and Y. Vemena. *Proof systems for Moss coalgebraic logic.* Theoretical Computer Science (2013), pp. 36-60. - ISSN 0304-3975

[Kur01] A. Kurz. *Coalgebræ and Modal Logic.* CWI, Amsterdam, Netherlands, 2001

[Kur00] A. Kurz. *Logics for coalgebræ and applications to computer science.* Dissertation thesis, Amsterdam, Netherlands, 2000

[Jac97] B. Jacobs, and J. Rutten. *A tutorial on (co) algebras and (co) induction.* Bulletin-European Association for Theoretical Computer Science, vol. 62, pp. 222-259, 1997.

[Mac45] S. Eilenberg, and S. Mac Lane. *General theory of natural equivalences.* Trans. Am. Math. Soc., vol. 58, pp. 231294, 1945.

[Mih12] D. Mihályi, V. Novitzká, and M. Ľaľová. *Intrusion Detection System Epistme.* Central European Journal of Computer Science. Vol. 2, no. 3 (2012), pp. 214-220. - ISSN 1896-1533

[Mih14] D. Mihályi, and V. Novitzká. *Towards to the Knowledge in Coalgebraic model IDS.* Computing and Informatics, 33, 1, pp. 61-78, 2014, ISSN 1335-9150

[Mih10] D. Mihályi, and V. Novitzká. *Princípy duality medzi konštruovaním a správaním programov.* Equilibria, Košice, 2010, (in slovak)

[Mos97] L. Moss. *Coalgebraic logic.* Annals of Pure and Applied Logic, Volume 99, Issues 13, Department of Mathematics, Indiana University, Bloomington, str. 241-259, USA, 1997

[Mra13] M. Mražík, and D. Mihályi. *Intrusion Calls Possibilities of Traceability for Running Functional Programm Based on Coalgebræ* . Electrical Engineering and Informatics 4 : proceedings of the Faculty of Electrical Engineering and Informatics of the Technical University of Košice, Košice, FEEI TU, 2013, pp. 796801.

[Per15] J. Perháč, and D. Mihályi. *Coalgebraic modeling of IDS behavior.* 2015 IEEE 13th International Scientific Conference on Informatics, November 18-20, 2015, Poprad, Slovakia, Danvers: IEEE, 2015, 201-205, DOI: 10.1109/Informatics.2015.7377833, ISBN 978-1-4673-9867-1.

[Slo11] V. Slodičák, and P. Macko. *Some new approaches in functional programming using algebras and coalgebræ* . Electronic Notes in Theoretical Computer Science, vol. 279, no. 3, pp. 4162, 2011.

[Mac11] V. Slodičák, and P. Macko. *The rle of linear logic in coalgebraical approach of computing.* Journal of Information and Organizational Sciences, vol. 35, no. 2, pp. 197213, 2011.

[Sma11] V. Slodičák, and P. Macko. *How to apply linear logic in coalgebraical approach of computing.* CECIIS-2011, 2011.

[Ste14] W. Steingartner, and D. Radakovic. *Categorical structures as expressing tool for differential calculus.* Central European Journal of Computer Science. Vol. 4, no. 3, p. 96-106. - ISSN 1896-1533, 2014.