CATEGORICAL SEMANTICS OF REFERENCE DATA TYPE

Daniel MIHÁLYI, Miloš LUKÁČ, Valerie NOVITZKÁ
Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics, Technical University of Košice,
Letná 9, 042 00 Košice, Slovak Republic, tel. +421 55 602 3175, e-mail: Daniel.Mihalyi@tuke.sk, Milos.Lukac@student.tuke.sk,
Valerie.Novitzka@tuke.sk

ABSTRACT
Reference types are very useful structures enabling handling with dynamic memory. In this paper we extend categorical model of type system with reference data type. We illustrate our approach on simple functional programming language T-NBL extended with reference type. After constructing parametric algebraic specification we construct categorical model of reference type and we show how our results can be implemented in functional programming language Ocaml.

Keywords: *reference data type, algebraic specification, categorical model, functional programming*

1. INTRODUCTION

Programming languages deal with various data structures that can be classified and identified by their types. A type is defined [10] as a collection of data with the same structure, from basic types, e.g. integers, reals, booleans to Church's types as records, variants, etc. Together with types there should be defined a set of operations between them. There are many programming languages that support also reference data types. A reference [13] is a value defining an access point to particular memory location and it is frequently called indirect addressing. Operations over reference type enable to allocate some memory space for data, to extract (dereference) data stored in this location and to write (assign) a new value to a referenced location. Usage of reference types increases flexibility of used memory, it enables to work with data structures with dynamic length, e.g. linked lists, etc.

Our paper presents a new approach of categorical modeling of reference types. As a basis we use a simple Typed Number-Boolean Language (T-NBL) [13]. We extend known algebraic specification of this language by reference type and its operational specifications together with new equational axioms. Then we construct a model of this algebraic specification as a category of type representations where we concern mostly on modeling reference type operations.

2. T-NBL DEFINITION

As a basic language in this paper we use typed Number Boolean Language (T-NBL) as a simple functional language based on evaluation of arithmetic expressions. Its definition follows the language of boolean and arithmetic expressions from [13]. In this section we introduce syntax of T-NBL.

We consider only two basic types in T-NBL: `Bool` of boolean values and `Int` of integer numbers.

$$T \quad ::= \quad \texttt{Bool} \mid \texttt{Int} \tag{1}$$

A program is a term constructed by using the following production rule:

$$\begin{aligned} t \quad ::= \quad & \texttt{true} \mid \texttt{false} \mid \texttt{0} \mid \texttt{succ}\, t \mid \texttt{pred}\, t \mid \texttt{iszero}\, t \mid \\ & \mid \texttt{if}\, t\, \texttt{then}\, t\, \texttt{else}\, t \mid (t) \end{aligned} \tag{2}$$

The first three alternatives (`true`, `false`, `0`) are constants. The next three (`succ t`, `pred t`, `iszero t`) are terms constructed by means of unary operations successor, predecessor and testing operation, respectively. The following term (`if` t `then` t `else` t) is conditional term and the last alternative expresses bracketing of terms.

T-NBL has boolean and numerical values

$$\begin{aligned} v \quad &::= \quad \texttt{true} \mid \texttt{false} \mid nv \\ nv \quad &::= \quad \texttt{0} \mid \texttt{succ}\, nv \mid \texttt{pred}\, nv \end{aligned} \tag{3}$$

We note that this simple language has no variables, therefore numerical values can be expressed only by using finite number of applications `succ` and/or `pred` operations.

3. ALGEBRAIC SPECIFICATION OF T-NBL EXTENDED BY REFERENCE TYPE.

We extend the syntax of T-NBL with reference type and its operations. We add new type $Ref(T)$ into production rule for types:

$$T ::= \ldots \mid Ref(T) \tag{4}$$

We need to add also new forms of terms:

$$t ::= \ldots \mid \texttt{alloc}\, t \mid !t \mid \texttt{assign}\, t\; t \mid \texttt{Null} \tag{5}$$

The operation `alloc` serves for allocation of a memory cell, ! serves for dereferencing and `assign` serves for modification of allocated value. `Null` is empty reference.

In computer science, namely in algebraic specifications, types can be specified using algebraic specification of abstract data types [7]. An algebraic specification consists of a many-typed signature and a set of equational axioms. An algebraic specification is a tuple

$$Spec = (\Sigma, \mathscr{E}) \tag{6}$$

where

$$\Sigma = (\mathscr{T}, \mathscr{F}). \tag{7}$$

In this notation

- $\mathscr{T}$ is a finite set of types names $\sigma, \tau, \upsilon ...$,
- $\mathscr{F}$ is a finite set of operational specifications in the form

  $$f : \sigma_1, ..., \sigma_n \rightarrow \tau \tag{8}$$

  where $\sigma_1, ..., \sigma_n$ are input types and τ is an output type, for $n \in N$,
- $\mathscr{E}$ is a finite set of equational logic axioms $e_1, e_2, \dots, e_n$. They express properties of operational specifications.

In T-NBL we have two types: *Bool* and *Int*. Algebraic specifications for these types are well known from literature [7, 14, 15]. Let Σ_{Int} and Σ_{Bool} be the signatures of the types *Int* and *Bool*, respectively. In this section we construct the algebraic specification for reference type. First, we must introduce a special new type *Unit* with single value *unit*. This type has no operations, it is an elementary type with signature

$$\begin{array}{l} \Sigma_{Unit} = (\\ \mathscr{T} = \{Unit\}, \\ \mathscr{F} = \emptyset) \end{array} \tag{9}$$

Now we can construct signature for reference type as parametrized signature. Let T be a type of T-NBL. The signature of reference type has the following form

$$\Sigma = (\Sigma_T, \Sigma_{Ref(T)}) \tag{10}$$

where Σ_T is a signature of parameter T and $\Sigma_{Ref(T)}$ has the form

$$\begin{array}{l} \Sigma_{Ref(T)} = (\\ \quad \mathscr{T} = \{Ref(T)\}, \\ \quad \mathscr{F} = \{ \\ \quad\quad alloc : T \rightarrow Ref(T), \\ \quad\quad ! : Ref(T) \rightarrow T, \\ \quad\quad assign : (Ref(T))^T \rightarrow Unit \\ \quad\quad null : \rightarrow Ref(T), \\ \quad \} \\ ) \end{array} \tag{11}$$

Three operations can be applied on reference type. The first one is *alloc* that reserves a memory space for a value of type T. The second operation ! enables to obtain a value stored in referenced memory space and the third one *assign* serves for assigning a new value to referenced memory space. The notation $(Ref(T))^T$ is an abbreviation of the function type $T \rightarrow Ref\ (T)$. We note that the output type of this operation is $Unit$. *null* is a special constant expressing unassigned memory location.

Axioms for reference type are conditional equations in the form of $if - then$ expressions. Let $r, s : Ref(T)$ be the variables of reference type $Ref(T)$. Let the variables $u : T, v : T$ be of a type T. Then we can form the following conditional equations for reference data type

$$\begin{array}{l} \mathscr{E} = \{ \\ \quad u = !(alloc\ u) \\ \quad v = !assign(alloc\ u, v) \\ \quad if\ (r = alloc\ u)\ and\ (r = s)\ then\ !s = u \\ \quad if\ (r = alloc\ u)\ and\ (s = alloc\ r)\ then\ !s = u \\ \quad if\ (r = null)\ then\ !r = "error" \\ \quad if\ (r = null)\ then\ assign(r, v) = "error" \\ \} \end{array} \tag{12}$$

The first equation expresses relation between alocation and dereferentiation. The second equation expresses relation between assigning and dereferencing. The following two conditional equations denote the known property of aliasing and reference composition, respectively. The last two conditional equations illustrate errors arising from an attempt to dereference unallocated memory and to modify a value of unallocated memory.

4. CATEGORICAL MODEL OF T-NBL EXTENDED BY REFERENCE TYPE.

To construct a model of our algebraic specification we have to assign to type names their representations and to operational specifications their actual operations. According to [7] heterogeneous algebras are not suitable for programming languages with reference types. We construct a model for T-NBL as a category $TNBL_{RefT}(\Sigma)$ over the signature Σ. This category has as category objects type representations and as category morphisms real operations over Σ.

First of all, we have to assign type representations as sets of values to basic type names in Σ

$$\begin{array}{ll} Int & \mapsto \mathbb{I} \\ Bool & \mapsto \mathbb{B} \\ Unit & \mapsto \{*\} \end{array} \tag{13}$$

where $\mathbb{I}$ is the set of integer numbers, $\mathbb{B}$ is the set of boolean values. We represent the type name $Unit$ as a singleton $\{*\}$. This object is also the terminal object of our category. Initial object of this category is empty set $\emptyset$, i.e. empty type.

A representation of reference type name $Ref(T)$ is not so simple. Assume that $\mathbb{T}$ denotes a type representation of any particular type from T-NBL. It is not enough to assign to reference type name only a set $\mathbb{R}$ of possible references (some abstract memory addresses)

$$Ref(T) \mapsto \mathbb{R} \tag{14}$$

because every reference should be bounded to some concrete language type. Binding can be represented by binary products between the object $\mathbb{R}$ and particular object $\mathbb{T}$ together with corresponding projections. For instance, $\mathbb{R}\times\mathbb{I}$ expresses the type representation of references pointing to integers together with two projections π_1 and π_2. Similarly, we construct the binary products $\mathbb{R}\times\mathbb{B}$, $\mathbb{R}\times\mathbb{U}$, $\mathbb{R}\times\mathbb{R}$ and $\mathbb{R}\times\emptyset$ for other types of our language. Based on [2], if a category has binary products and terminal object it has finite products. Therefore we consider that our category has finite products including empty product denoted by singleton $\{*\}$.

To handle with references of different types in unified way we use coproducts of objects. For example, the object $\mathbb{B}+\mathbb{I}$ is a coproduct of boolean and integer type representations. An element of this coproduct type is either integer number or boolean value according to their origins defined by coprojections κ_1,κ_2. Based on [4], if a category has binary coproducts and initial object, it has finite coproducts. Therefore we assign to reference type name $Ref(T)$ the following coproducts

$$Ref(T)\mapsto(\mathbb{R}\times\mathbb{I})+(\mathbb{R}\times\mathbb{B})+(\mathbb{R}\times\mathbb{U})+(\mathbb{R}\times\mathbb{R})+(\mathbb{R}\times\emptyset) \tag{15}$$

Using distributive laws

$$\begin{aligned}(Z\times X)+(Z\times Y)&\cong Z\times(X+Y)\\ (X\times\emptyset)&\cong\emptyset\end{aligned} \tag{16}$$

where X,Y and Z are category objects, we can write

$$Ref(T)\mapsto\mathbb{R}\times\mathbb{T} \tag{17}$$

where

$$\mathbb{T}=\mathbb{B}+\mathbb{I}+\mathbb{U}+\mathbb{R}+\emptyset \tag{18}$$

We have to assign representations to the operational specification of the signature Σ in (10). To model the operation *assign* we need that our category has exponent objects. Because objects of our category are sets, from [1] the category $TNBL_{RefT}(\Sigma)$ has exponent objects, i.e. for any objects X and Y it has an object Y^X as a set of morphisms from X to Y.

According to discussion above our categorical model has finite products and finite coproducts, exponential objects together with distributive property (16). This model is illustrated in Fig. 1.

Now we can construct operations over Σ as category morphisms. The operations of predecessor/successor ($pred/succ$) are modeled as endomorphisms

$$\begin{aligned}&[\![pred]\!]:\mathbb{I}\to\mathbb{I}\\ &[\![succ]\!]:\mathbb{I}\to\mathbb{I}\end{aligned} \tag{19}$$

where

$$\begin{aligned}&[\![pred]\!](i)=i-1\\ &[\![succ]\!](i)=i+1\end{aligned} \tag{20}$$

for $i\in\mathbb{I}$. The operation $[\![iszero]\!]$ is a morphism from $\mathbb{I}$ to $\mathbb{B}$, where

$$[\![iszero]\!]:\mathbb{I}\to\mathbb{B} \tag{21}$$

$$[\![iszero]\!](i)=\begin{cases} true, & \text{if } i=0\\ false, & \text{otherwise}\end{cases} \tag{22}$$

The conditional operation $[\![if]\!]$ is modeled as the morphism

$$[\![if]\!]:\mathbb{B}\times\mathbb{T}\times\mathbb{T}\to\mathbb{T} \tag{23}$$

and defined by

$$[\![if]\!]([\![t_1]\!],[\![t_2]\!],[\![t_3]\!])=\begin{cases}[\![t_2]\!], & \text{if } [\![t_1]\!]=true\\ [\![t_3]\!], & \text{if } [\![t_1]\!]=false\end{cases} \tag{24}$$

where $[\![t_i]\!]$, for $i=1,\dots,3$ are values of corresponding terms, $[\![t_1]\!]\in\mathbb{B}$ and $[\![t_2]\!],[\![t_3]\!]$ are values of the coproduct object $\mathbb{T}$ (from (18)) constructed by the same coprojection.

Now we model operations for reference type specifications introduced in (11). Let $[\![t]\!]$ be a value of a term t of a type T.

The operation *alloc* is modeled as a morphism

$$[\![alloc]\!]:\mathbb{T}\to\mathbb{R}\times\mathbb{T} \tag{25}$$

defined by

$$[\![alloc]\!]([\![t]\!])=\langle r,[\![t]\!]\rangle \tag{26}$$

where $[\![alloc]\!]([\![t]\!])$ assigns to a value $[\![t]\!]$ an element $\langle r,[\![t]\!]\rangle$ of product type $\mathbb{R}\times\mathbb{T}$. This morphism binds a value $[\![t]\!]$ with abstract location r.

The operation ! is modeled as a morphism

$$[\![!]\!]:\mathbb{R}\times\mathbb{T}\to\mathbb{T} \tag{27}$$

defined by

$$[\![!]\!](\langle r,[\![t]\!]\rangle)=[\![t]\!] \tag{28}$$

where $[\![!]\!]$ takes as an argument abstract location with it's value $\langle r,[\![t]\!]\rangle$ and returns the value $[\![t]\!]$ stored there.

We model the operation *assign* as a morphism to final object

$$[\![assign]\!]:(\mathbb{R}\times\mathbb{T})^{\mathbb{T}}\to\{*\} \tag{29}$$

defined by

$$[\![assign]\!]([\![t]\!]\mapsto\langle r,[\![t]\!]\rangle)=* \tag{30}$$

This operation modifies a value stored on a location r by the value of $[\![t]\!]$. The result $*$ of this operation raises that the modification of this value was successful.

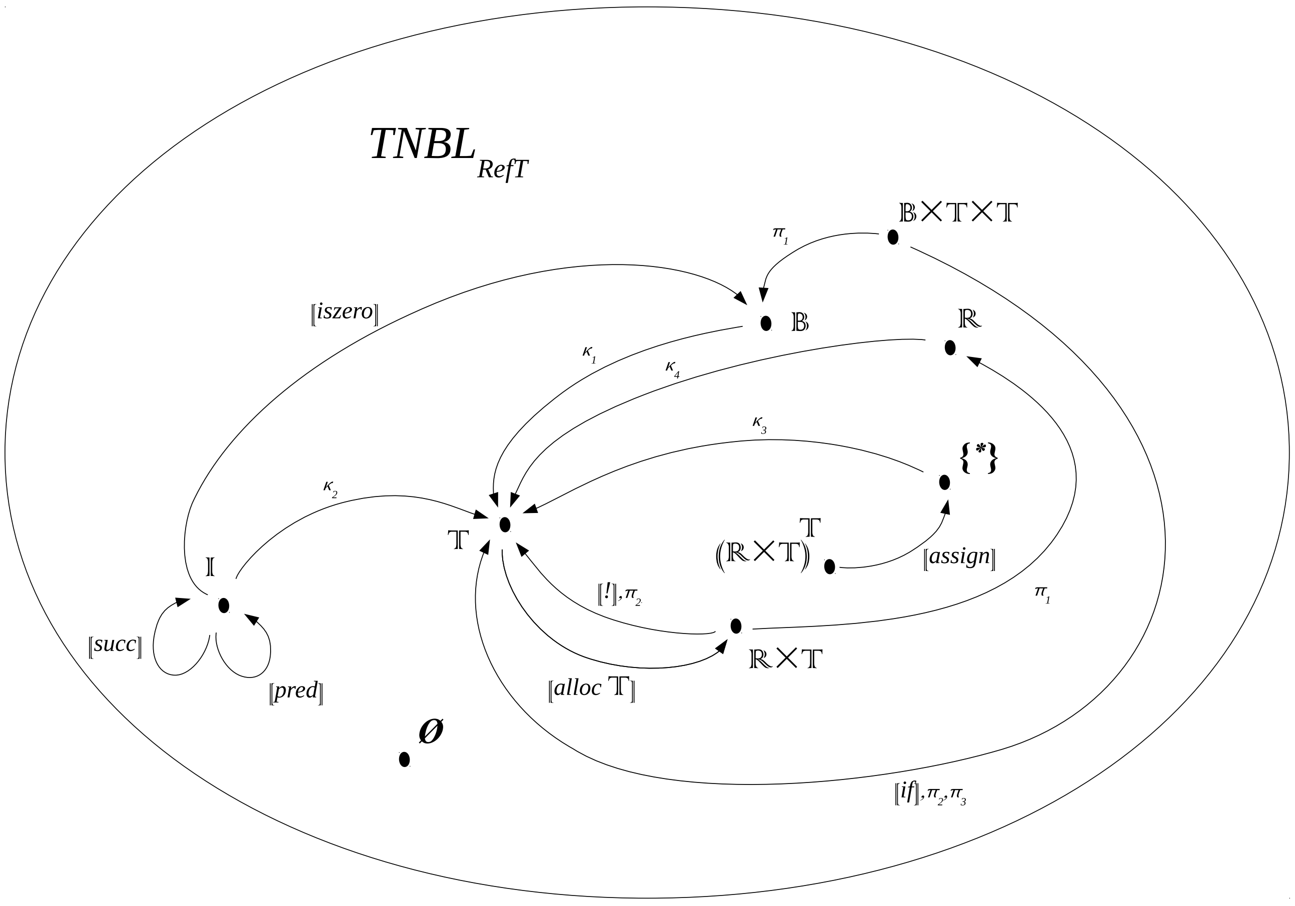

Fig. 1 Categorical model T-NBL+Ref

5. BEHAVIOR OF REFERENCE TYPE IN FUNCTIONAL LANGUAGE

Reference type is often used in many programming languages in various programming paradigms. In this section we illustrate behavior of reference type in functional object oriented language *Ocaml* [11]. In this paper we use Ocaml interpreter enabling immediate execution of particular program and rapid response. First, we declare the reference type as:

```
type 'a pointer = Null | Pointer of 'a ref;;
```

In this program line we declare the possible forms of reference type: empty reference (`Null`) or some address in computer memory.

In the following lines we define three operations on reference according to signature in (11).

```
let ( ! ) = function
    | Null -> invalid_arg "Attempt to
  dereference the null pointer"
    | Pointer r -> !r;;

let assign p v =
    match p with
    | Null -> invalid_arg "Attempt to
  assign the null pointer"
    | Pointer r -> r := v;;

let alloc x = Pointer (ref x);;
```

First of these operations is `!`, dereference, that returns a content of a memory cell. The operation `assign` modifies a content of it. The last operation `alloc` reserves new memory cell and it stores the value of `x` into this place.

The following program lines show the properties of reference type defined in (12). The symbol "#" in the front of a line denotes input to the interpreter and its response is denoted by the symbol "-" in the front of the line.

```
1  # let u = alloc 5 in !u;;
2  - : int = 5
3  #
4  # let u = alloc 5 in let v = 6 in assign u v; !u;;
5  - : int = 6
6  #
7  # let r = alloc 5 in let t = r in !t;;
8  - : int = 5
9  #
10 # let r = alloc 5;;
11 val r : int pointer = Pointer {contents = 5}
12 # let t = alloc r;;
13 val t : int pointer pointer = Pointer {contents
```

```
   = Pointer {contents = 5}}
14 # !t;;
15 - : int pointer = Pointer {contents = 5}
16 #
17 # alloc Null;;
18 - : '_a pointer pointer
   = Pointer {contents = Null}
19 #
20 # !Null;;
21 Exception: Invalid_argument "Attempt
   to dereference the null pointer".
22 #
23 # assign Null 1;;
24 Exception: Invalid_argument "Attempt
   to assign the null pointer".
25 #
```

Line 1 corresponds to the first axiom in (12) defining relation between allocating and dereferencing. Line 4 shows how an allocated value can be modified by `assign` operation. Line 7 corresponds with the third axiom about aliasing. The lines from 10 to 15 express composition of references. The remaining lines illustrate error situations arising in an attempt to dereference empty reference `Null` and in assigning new value to empty reference.

All the terms interpreted above by Ocaml are modeled as path, morphism compositions, in our category model of the language T-NBL+Ref in Fig. 1.

6. CONCLUSION

Reference types are often used in various paradigms of programming. In our paper we present how references can be modelled as a category. First, we extend a simple typed language T-NBL with reference type, then we construct algebraic specification as parametric signature together with conditional equational axioms. We construct categorical model of so extended T-NBL using distributive category of types. In the last section we illustrate conditional equational axioms of the algebraic specification of extended T-NBL on examples in functional programming language Ocaml. Our results can serve for generalization our approach to model typed lambda-calculus extended with reference types in categorical terms.

ACKNOWLEDGEMENT

This paper was supported by two projects:
Slovak Research and Development Agency under the contract No. APVV-0008-10 "Modelling, simulation and implementation of GPGPU-enabled architectures of high-throughput network security tools."

Agency of the Ministry of Education, Science, Research and Sport of the Slovak Republic for Structural Founds of EU, Project ITMS code 26110230093: BIP4

REFERENCES

[1] AWODEY, S.: Category Theory, Oxford Logic Guides, OUP Oxford, 2010, ISBN 9780199587360.

[2] BARR, M. – WELLS, Ch.: Category Theory for Computing Science, Prentice Hall International (UK) Ltd., 1990, ISBN 0-13-120486-6.

[3] BIRD, R. – WALDER, P.: Introduction to Functional Programming, Programming Research Group, Oxford University, Department of Computer Science, University of Glasgow, 1988.

[4] BLUTE, R. – SCOTT, P.: Category Theory for Linear Logicians, Linear Logic Summer School, Cambridge University Press, 2003.

[5] CHAILLOUX, E. – MANOURY, P. - PAGANO, B.: Developing Applications with Objective Caml, O'REILLY, Paris, 2000.

[6] CROLE, R. L.: Categories for Types, Cambridge University Press, 1993, ISBN 9780521457019.

[7] EHRIG, H. – MAHR, B.: Fundamentals of Algebraic Specification 1: Equations and Initial Semantics, Springer-Verlag, New York, 1985, ISBN 3-540-13718-1.

[8] GIBBONS, J.: Conditionals in Distributive Categories, Technical report CMS-TR-97-01, School of Computing and Mathematical Sciences, Oxford Brookes University, 1996.

[9] MIHÁLYI, D. – NOVITZKÁ, V.: Princípy duality medzi konštruovaním a správaním programov, Equilibria, Košice, 2010. ISBN 978-80-89284-58-0.

[10] NOVITZKÁ, V.: *Church's Types in Logical Reasoning on Programming*, Acta Electrotechnica at Informatica **6**, No. 2 (2006) 27–31.

[11] LEROY, X. et al: The OCaml System Release 4.01, Documentation and Users Manual, 2013 `http://caml.inria.fr/pub/docs/manual-ocaml/`.

[12] NOVITZKÁ, V. – SLODIČÁK, V.: Kategorické štruktúry a ich aplikácie v informatike, Equilibria, Košice, 2010, ISBN 978-80-89284-67-2.

[13] PIERCE, B. C.: Types and Programming Languages, The MIT Press, 2002, ISBN 0-262-16209-1.

[14] SANNELLA, D. – TARLECKI A.: *Essential Concepts of Algebraic Specification and Program Development*, Formal Aspects of Computing **9**, No. 3 (1997) 229–269

[15] WIRSING, M.: Algebraic Specification, In (Jan van Leeuwen, ed.) Handbook of Theoretical Computer Science, Elsevier, 1990, pp. 675–788.

Received November 12, 2013, accepted December 24, 2013

BIOGRAPHY

Miloš Lukáč defended his M.Sc. thesis in 2013. He is concerned with type theory, category theory and functional programming.

Daniel Mihályi has worked as a researcher at the Department of Computers and Informatics of the Faculty of Electrical Engineering and Informatics at Technical University in Košice and later as Assistant Professor. In 2009 he defended PhD. Thesis ”Duality Between Formal Description of Program Construction and Program Behaviour”. The main area of his research includes applications of category theory in informatics and using source-based logical systems for formal description of program systems behavior.

Valerie Novitzká defended her PhD. Thesis ”On Semantics of Specification Languages” at Hungarian Academy of Sciences in 1989. She has worked at Department of Computers and Informatics since 1998, firstly as Assistent Professor, from 2004 as Associated Professor and since 2008 as Full Professor. Her research area covers category theory, categorical logic, type theory, classical and linear logic and theoretical foundations of program development.