Simple-typed Functional Language Modeled by Category Theory

Ján Perháč[1][0000−0001−6347−2409], Zuzana Bilanová[1][0000−0002−0128−5111], and Gabriela Havrilčáková[2][0000−0002−3788−2217]

[1] Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics, Technical University of Košice, Letná 9 042 00 Košice, Slovak Republic
{jan.perhac,zuzana.bilanova}@tuke.sk
[2] gabriela.havrilcakova@student.tuke.sk

Abstract. In this paper we present the simply-typed Lambda calculus extended by typed arithmetic expressions, Boolean values, derived forms, and reference types. We briefly present its syntax and semantics. Based on that we construct a model of this language using category theory.

Keywords: Category theory · Simply-typed Lambda calculus · Type theory.

1 Introduction

The simply-typed Lambda calculus is considered to be the simplest functional programming language. It was introduced by the Alonzo Church and Stephen Kleen [1]. In our work, we use a language extended by typed arithmetic expressions, Boolean values, derived forms, and reference types. The goal of this paper is to unify the individual extensions of simply-typed Lambda calculus to one syntax, one many-typed signature, and one algebraic specification. Based on that, we construct its corresponding model by category theory.

In our previous research, we have worked with type theory and linear logic [2] [3] [4] where we have published our logical extensions of it. We have implemented our extensions into teaching of type theory course in master level with interactive web portal [5]. Our portal also include questionnaires for students. Based on that we are trying to improve a course.

2 Simply typed λ-calculus and T-NBL definition

λ-calculus with The typed Number Boolean Language (T-NBL) is the base language of this paper. As the basic sets of typed expressions of the λ-calculus are considered the sets of natural numbers *Nat* and boolean values *Bool* [6], [7].

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

Definition of simple typed λ-calculus with functional-type and with typed Number Boolean Language (T-NBL) is composed of semantics and syntax [8] with the following syntax:

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

The first four alternatives of the BNF production rule (2) belong to λ-calculus and the others to T-NBL. Signature is define by applying the syntax of operations and types. It is composed of pairs of sets [9]:

$$\Sigma = (\mathcal{T}, \mathcal{F}) \tag{3}$$

- $\mathcal{T} = \{\texttt{Bool}, \texttt{Nat}, T\}$ represents a finite set of the names of basic types.
- $\mathcal{F} = \{\texttt{x}, \texttt{abs}, \texttt{app}, \texttt{true}, \texttt{false}, \texttt{if}, \texttt{0}, \texttt{succ}, \texttt{pred}, \texttt{iszero}\}$ represents a finite set of the specification of operations.

Signature has a pattern $\Sigma_{T-NBL \ + \ \lambda k}$ and the following form:

$$\begin{aligned} & \Sigma_{T-NBL \ + \ \lambda k} = (\\ & \mathcal{T} = \{Bool, Nat, T\}, \\ & \mathcal{F}_{T-NBL \ + \ \lambda K} = \{ \\ & true, false :\rightarrow Bool, \\ & 0 :\rightarrow Nat, \\ & succ : Nat \rightarrow Nat, \\ & pred : Nat \rightarrow Nat, \\ & iszero : Nat \rightarrow Bool, \\ & if : Bool, T, T \rightarrow T, \\ & x :\rightarrow T, \\ & abs : T \rightarrow T, \\ & app : (T \rightarrow T), T \rightarrow T, \}) \end{aligned} \tag{4}$$

Model of algebraic specification is characterized by [10]:

1. **Type assignment** - to every type name $T \in \mathcal{T}$ from signature Σ is assigned the appropriate type representation $[\![\ T]\!]$,
2. **Real operation assignment** - to every specification of operation:

$$f : T_1, \ldots, T_n \rightarrow T_0 \tag{5}$$

is assigned a real appropriate operation:

$$[\![f]\!] : [\![T_1]\!] \times \ldots \times [\![T_n]\!] \rightarrow [\![\ T_0]\!]. \tag{6}$$

Categorical model is created in the way that, to every name of basic types (`Nat` or `Bool`) is assigned a set of real types ($\mathbb{N}$ or $\mathbb{B}$) and also to the operation specification, for example, to the `pred` is assigned a real operation $[\![\texttt{pred}]\!]$. Real operation works with semantic domens, on which is arity defined by cartesian product.

Categorical model consists of objects and morphisms. Objects are considered as semantical domens $\mathbb{N}$, $\mathbb{B}$, final object 0, and the initial object $\{^*\}$. Morphisms are composed of operations as follows:

– the morphisms of the T-NBL language:
  - $[\![\texttt{true/false}]\!] : \{^*\} \to \mathbb{B}$ defines the constants `true` or `false`.
  - $[\![\texttt{0}]\!] : \{^*\} \to \mathbb{N}$ defines the constant `0`.
  - Operations successor and predecessor are represented in the model as

$$[\![\texttt{succ}, \texttt{pred}]\!] : \mathbb{N} \to \mathbb{N}. \tag{7}$$

  - Operation `iszero` is used for verification of zero value, is represented as

$$[\![\texttt{iszero}]\!] : \mathbb{N} \to \mathbb{B}. \tag{8}$$

  - The conditional operation `if` is represented as morphism

$$[\![\texttt{if}]\!] : \mathbb{B} \times (\mathbb{B} \ + \ \mathbb{N}) \times (\mathbb{B} \ + \ \mathbb{N}) \to \mathbb{B} \ + \ \mathbb{N}. \tag{9}$$

– the morphisms of the λ-calculus:
  - $[\![\texttt{x}]\!] : \{^*\} \to \mathbb{B} \ + \ \mathbb{N}$ morphism for expression of variable.
  - $[\![\texttt{abs}]\!] : (\mathbb{B} \ + \ \mathbb{N}) \to (\mathbb{B} \ + \ \mathbb{N})$ morphism for expression of abstraction.
  - $[\![\texttt{app}]\!] : (\mathbb{B} \ + \ \mathbb{N})^{(\mathbb{B} \ + \ \mathbb{N})} \times (\mathbb{B} \ + \ \mathbb{N}) \to (\mathbb{B} \ + \ \mathbb{N})$ morphism for expression of application.

Model contains coprojections κ_1, κ_2 and projections π_1, π_2, φ_1, φ_2, φ_3.

The figure 1 shows categorical model of the T-NBL (purple color) with the λ-calculus (orange color).

3 Derived forms definition

Some of the programming languages have side effects, for that reason derived forms are used [11]. Derived forms ensures, that abbreviations are established to λ-terms, determines the evaluation approach of λ-terms, simplifying the writing and making the code more transparent. Every derived shape can be determined with basic forms of λ-calculus, that are sequencing, wildcards, ascription and let binding. We add a singleton set to basic sets of typed expressions on derived forms [10].

$$T ::= \ldots \mid \texttt{Unit} \tag{10}$$

Syntax is extended by these alternatives [8]:

$$t ::= \ldots \mid (\lambda x : \texttt{Unit}.t)t \mid (\lambda\_ : T.x)t \mid t \ as \ T \mid (\lambda x : T.t), \tag{11}$$

where the first alternative is sequence (seq), the second is wildcards (wild), the third is ascription (asc) and the last one represents let binding (let).

Signature of derived forms consists of following:

– $\mathcal{T} = \{\texttt{Bool}, \texttt{Nat}, \texttt{Unit}, T\}$,
– $\mathcal{F} = \{\texttt{seq}, \texttt{wild}, \texttt{asc}, \texttt{let}\}$.

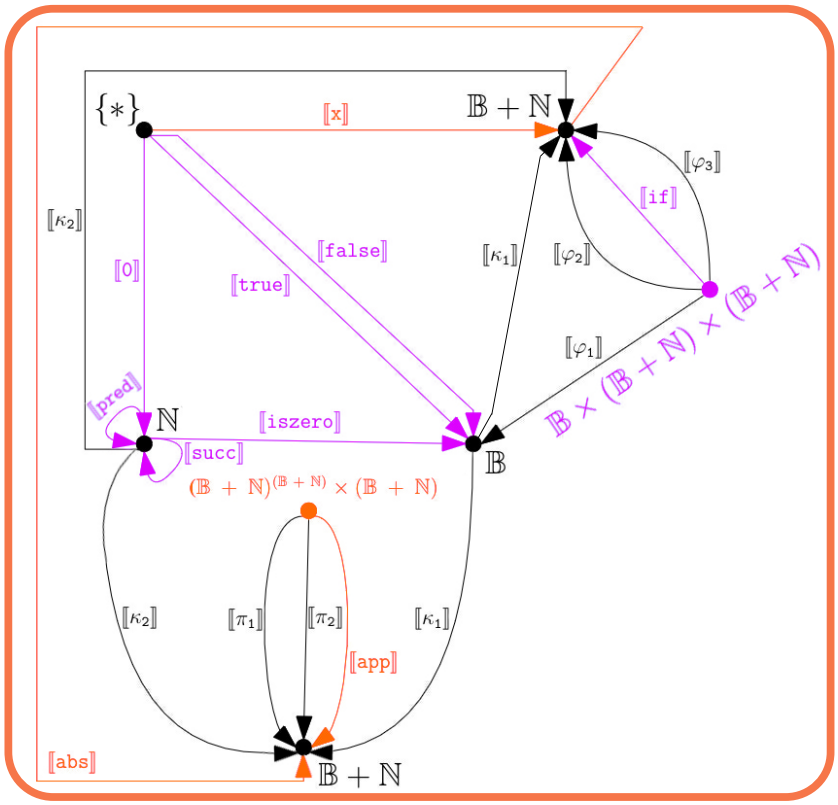

Fig. 1. Categorical model of T-NBL and simply typed λ-calculus

And its form is:

$$\begin{aligned}
&\Sigma_{DS} = (\\
&\mathcal{T} = \{Bool, Nat, Unit, T\},\\
&\mathcal{F}_{DS} = \{\\
&seq :\rightarrow Unit, T \rightarrow T,\\
&wild : T \rightarrow T,\\
&asc :\rightarrow T,\\
&let : T, T \rightarrow T\})
\end{aligned} \tag{12}$$

Categorical model of derived forms consists of objects, with these semantic domens $\mathbb{N}$, $\mathbb{B}$, $\mathbb{U}$, final object 0, and initial object $\{*\}$. Morphisms consists of operations of derived forms as follows:

- Operation sequence is represented as morphism $[\![\texttt{seq}]\!] : (\mathbb{U} \times (\mathbb{B} + \mathbb{N} + \mathbb{U})) \rightarrow \mathbb{B} + \mathbb{N} + \mathbb{U}$.
- Wildcards are defined as $[\![\texttt{wild}]\!] : \mathbb{B} + \mathbb{N} + \mathbb{U} \rightarrow \mathbb{B} + \mathbb{N} + \mathbb{U}$.
- Ascription represents morphism $[\![\texttt{asc}]\!] : \{*\} \rightarrow \mathbb{B} + \mathbb{N} + \mathbb{U}$,
- Let binding is $[\![\texttt{let}]\!] : ((\mathbb{B} + \mathbb{N} + \mathbb{U}) \times (\mathbb{B} + \mathbb{N} + \mathbb{U})) \rightarrow \mathbb{B} + \mathbb{N} + \mathbb{U}$.

Model contains coprojections κ_1, κ_2, κ_3 and projections π_1, π_2, φ_1, φ_2.

Corresponding categorical model of derived forms is depicted in the figure 2.

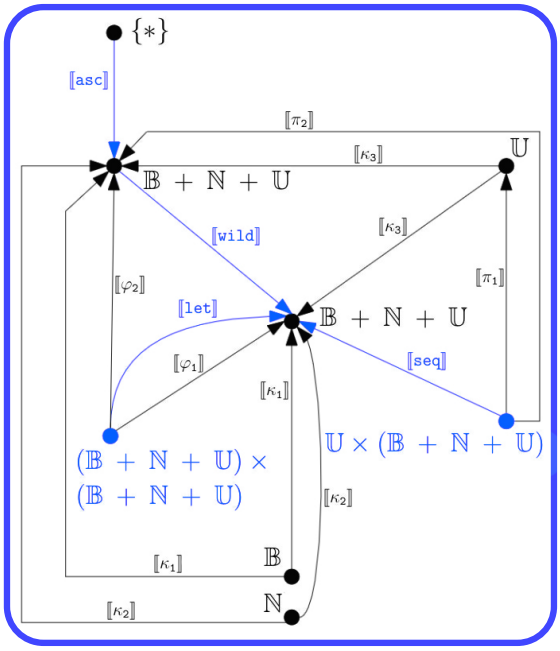

Fig. 2. Categorical model of derived forms

4 Reference type definition

In the previous sections we have mentioned just pure elements, but most programming languages contains also inpure elements. That is the reason for some of the side effects, and for that referentions are used. Referention determines a value, that contains an adress of another value. Reference types are types, that use just as much memory, as much is needed for the calculation. The basic operations of reference types are alloc, dereference and assign. We add a reference type to the basic sets of typed expressions [12], [13].

$$T ::= \ldots \mid \texttt{Ref} \tag{13}$$

Syntax is extended by these alternatives [8]:

$$t ::= \ldots \mid \texttt{ref}\ t \mid !t \mid t\ :=\ t, \tag{14}$$

where the first alternative is alloc (ref), the second is dereference (!) and the last one represents assign (:=).

Signature of reference types consists of:

- $\mathcal{T} = \{\texttt{Bool}, \texttt{Nat}, \texttt{Unit}, \texttt{Ref}, T\}$,
- $\mathcal{F} = \{\texttt{ref}, !, :=\}$.

And has a form:

$$\begin{array}{l} \Sigma_{Ref} = (\\ \mathcal{T} = \{Bool, Nat, Unit, T\}, \\ \mathcal{F}_{REF} = \{ \\ ref : T \to Ref\ T, \\ ! : Ref\ T \to T, \\ :=\ \ : (Ref\ T\ ^{T}) \to Unit, \}) \end{array} \tag{15}$$

Categorical model of reference types is composed of objects [14], [15], where belong these semantic domens $\mathbb{N}$, $\mathbb{B}$, $\mathbb{R}$, $\mathbb{U}$, final object 0, and beginning object {*}. Morphisms contains following operations:

- alloc $[\![\texttt{ref}]\!] : (\mathbb{B} + \mathbb{N} + \mathbb{R} + \mathbb{U}) \to \mathbb{R} \times (\mathbb{B} + \mathbb{N} + \mathbb{R} + \mathbb{U})$,
- dereference $[\![!]\!] : \mathbb{R} \times (\mathbb{B} + \mathbb{N} + \mathbb{R} + \mathbb{U}) \to \mathbb{B} + \mathbb{N} + \mathbb{R} + \mathbb{U}$,
- assign $[\![:=]\!] : (\mathbb{B} + \mathbb{N} + \mathbb{R} + \mathbb{U}) \times (\mathbb{R} \times (\mathbb{B} + \mathbb{N} + \mathbb{R} + \mathbb{U})) \to \mathbb{U}$.

Model contains coprojections κ_1, κ_2, κ_3, κ_4 and projections π_1, π_2, φ_1, φ_2. The corresponding categorical model of reference type is depicted in the figure 3.

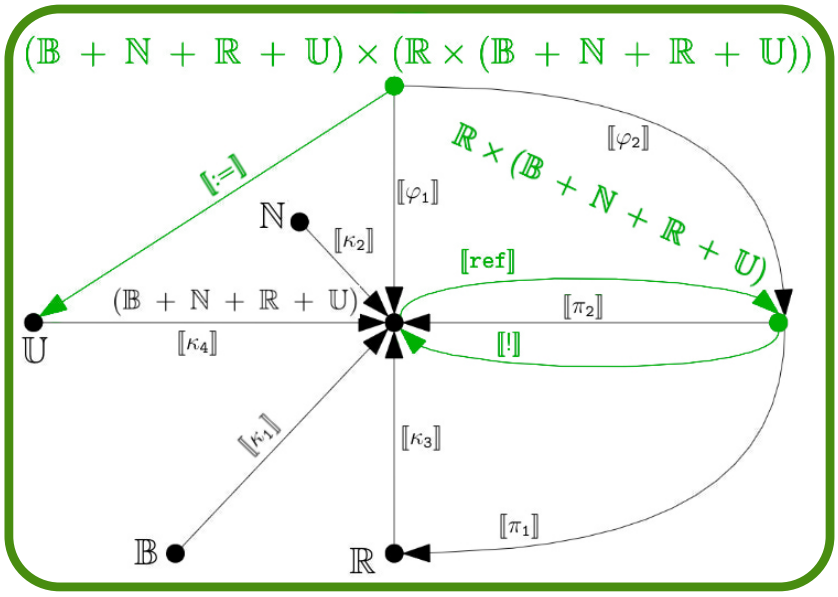

Fig. 3. Categorical model of reference type

5 Simply typed λ-calculus extended by T-NBL,derived forms and reference type

In this paper, we introduce the simply typed λ-calculus extended by T-NBL, derived forms and reference type.

Basic sets of typed expressions are as follows:

$$T ::= \texttt{Bool} \mid \texttt{Nat} \mid \texttt{Ref} \mid \texttt{Unit}. \tag{16}$$

The syntax of our language is expressed by the following BNF production rule:

$$\begin{array}{rl} t ::= & \texttt{true} \mid \texttt{false} \mid \texttt{if } t \texttt{ then } t \texttt{ else } t \mid 0 \mid \texttt{succ } t \mid \texttt{pred } t \mid \texttt{iszero } t \mid (t) \\ & \mid x \mid \lambda x : T.\ t \mid t\ t \mid \\ & \mid \texttt{ref } t \mid !t \mid t := t \mid \\ & \mid (\lambda x : \texttt{Unit}.t)t \mid (\lambda\_ : T.x)t \mid t\ as\ T \mid (\lambda x : T.t). \end{array} \tag{17}$$

Signature of extended simply typed λ-calculus contains:

- $\mathcal{T} = \{\texttt{Bool}, \texttt{Nat}, \texttt{Unit}, \texttt{Ref}, T\}$,
- $\mathcal{F} = \{$union of sets $\mathcal{F}_{T-NBL}, \mathcal{F}_{\lambda K}, \mathcal{F}_{REF}\ a\ \mathcal{F}_{DS}\}$.

  Where:
  - $\mathcal{F}_{T-NBL} = \{\texttt{true}, \texttt{false}, \texttt{0}, \texttt{succ}, \texttt{pred}, \texttt{iszero}, \texttt{if}\}$,
  - $\mathcal{F}_{\lambda K} = \{\texttt{x}, \texttt{abs}, \texttt{app}\}$,
  - $\mathcal{F}_{REF} = \{\texttt{ref}, !, :=\}$,
  - $\mathcal{F}_{OT} = \{\texttt{seq}, \texttt{wild}, \texttt{asc}, \texttt{let}\}$.

Signature has a form:

$$\begin{array}{l} \Sigma_{\lambda+TNBL+OT+Ref} = (\\ \mathcal{T} = \{Bool, Nat, Unit, T\}, \\ \mathcal{F} = \{ \\ true, false :\rightarrow Bool, \\ 0 :\rightarrow Nat, \\ succ : Nat \rightarrow Nat, \\ pred : Nat \rightarrow Nat, \\ iszero : Nat \rightarrow Bool, \\ if : Bool, T, T \rightarrow T, \\ x : \ \rightarrow T, \\ abs : T \rightarrow T, \\ app : (T \rightarrow T), T \rightarrow T, \\ ref : T \rightarrow Ref\ T, \\ ! : Ref\ T \rightarrow T, \\ := \ : (Ref\ T^T) \rightarrow Unit, \\ seq : Unit, T \rightarrow T, \\ wild : T \rightarrow T, \\ asc :\rightarrow T, \\ let : T, T \rightarrow T\}) \end{array} \tag{18}$$

The categorical model of typed λ-calculus extended by T-NBL, reference and derived forms was created by merging of the categorical models from past sections. It also contains following

- coprojections κ_1, κ_2, κ_3, κ_4, and
- projections π_1, π_2, π_3, φ_1, φ_2, β_1, β_2, γ_1, γ_2, δ_1, δ_2, ε_1, ε_2.

Categorical model is depicted in the next figure 4, where the separated extensions are represented by different colors. Purple is for the extension of T-NBL, orange stands for λ-calculus, green is for reference and blue represents the extension of derived forms.

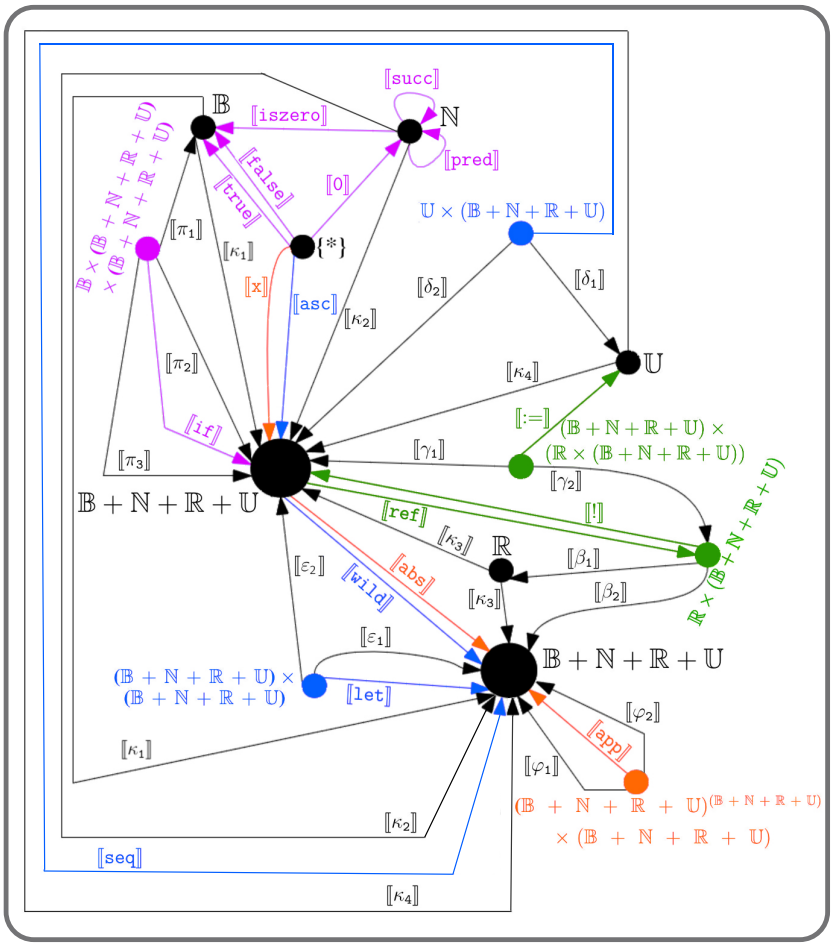

Fig. 4. Categorical model of extended λ-calculus

6 Conclusions

In this paper, we have introduced the simply typed λ-calculus extended by T-NBL, derived forms, and reference types. The main result is a unified syntax, signature, algebraic model, and categorical model of simply typed λ-calculus extended by T-NBL, reference types, and derived forms. Final categorical model contains all the extensions of the simply typed λ-calculus, they are separated by different colors for better transparency of the model.

Acknowledgment

This work was supported by the following projects:

- Faculty of Electrical Engineering and Informatics, Technical University of Košice under the contract No. FEI-2018-59: Semantic Machine of Source-Oriented Transparent Intensional Logic.
- Slovak Research and Development Agency under the contract No. SK-AT-2017-0012: Semantics technologies for computer science education.

References

1. A. Church, "A formulation of the simple theory of types," *The journal of symbolic logic*, vol. 5, no. 2, pp. 56–68, 1940.
2. J. Perháč and D. Mihályi, "Intrusion detection system behavior as resource-oriented formula," *Acta Electrotechnica et Informatica*, vol. 15, no. 3, pp. 9–13, 2015.
3. L. Vokorokos, E. Danková, and N. Ádám, "Task scheduling in distributed system for photorealistic rendering," in *2010 IEEE 8th International Symposium on Applied Machine Intelligence and Informatics (SAMI)*. IEEE, 2010, pp. 43–47.
4. J. Juhár and L. Vokorokos, "Separation of concerns and concern granularity in source code," in *2015 IEEE 13th International Scientific Conference on Informatics*. IEEE, 2015, pp. 139–144.
5. D. Mihályi, M. Peniašková, J. Perháč, and J. Mihelič, "Web based questionnaires for type theory course," *Acta Electrotechnica et Informatica*, 2017, (draft).
6. D. Mihályi and V. Novitzká, *Type theory*. Technical University of Košice, 2015.
7. V. Novitzká and A. Verbová, "Dependent types in mathematical theory of programming."
8. B. C. Pierce and C. Benjamin, *Types and programming languages*. MIT press, 2002.
9. H. Reichel, *Initial computability, algebraic specifications, and partial algebras*. Oxford University Press, Inc., 1987.
10. D. Mihályi, M. Lukáč, and V. Novitzká, "Categorical semantics of reference data type," *Acta Electrotechnica et Informatica*, vol. 13, no. 4, pp. 64–69, 2013.
11. P. B. Andrews, *An introduction to mathematical logic and type theory*. Springer Science & Business Media, 2002, vol. 27.
12. E. Chovancová, N. Adám, A. Baláž, E. Pietriková, P. Fecil'ak, S. Šimoňák, and M. Chovanec, "Securing distributed computer systems using an advanced sophisticated hybrid honeypot technology," *Computing and Informatics*, vol. 36, no. 1, pp. 113–139, 2017.

13. V. Novitzká and D. Mihályi, "Polymorphic type theory as a base for categorical logic," *Acta Electrotechnica et Informatica No*, vol. 7, no. 1, p. 3, 2007.
14. T. Hagino, "A typed lambda calculus with categorical type constructors," in *Category theory and computer science*. Springer, 1987, pp. 140–157.
15. L. Vokorokos, A. Baláž, and M. Chovanec, "Distributed detection system of security intrusions based on partially ordered events and patterns," in *Towards Intelligent Engineering and Information Technology*. Springer, 2009, pp. 389–403.