Computer Science Journal of Moldova, vol.**26**, no.1(76), 2018

Implementation of the Composition-nominative Approach to Program Formalization in Mizar

Ievgen Ivanov, Artur Korniłowicz, Mykola Nikitchenko

Abstract

In this paper we describe an ongoing work on implementation of the composition-nominative approach to program formalization in Mizar proof assistant based on the first-order logic and axiomatic set theory. The further aim of this work is development of a formal verification tool for software which processes and communicates with complex forms of data.

Keywords: Formal methods, program semantics, semistructured data, formalization, proof assistant.

1 Introduction

Formal verification of software systems has been a topic of interest of researches in computer science for more than fifty years. During this period many formal software verification tools based on different theoretical frameworks (automata theory, first-order logic, dynamic logics, program logics, etc.) were developed, but most of the existing tools are still in research stage and their usage in software industry is negligible. Some reasons include:

- they do not integrate well into typical software development cycles;

- successful practical application of such tools requires specialized knowledge, is labour intensive, time consuming, and not cost effective for most software projects.

However, in industries related to development of safety-critical systems such as aerospace, automotive, health technology formal verification of software plays a more significant, but still limited role.

The well known tools that support or aid formal verification of software for safety-critical systems include:

- model checkers such as Simulink Design Verifier, Systerel Smart Solver;

- verified translators and compilers such as CompCert that generate code in a low-level language or machine code from the source code in a high-level language that is proven to be equivalent to the source code under the assumptions of the formalized source and target language semantics;

- microkernels and hypervisors such as seL4 and CertiKOS that are formally verified with respect to formal specifications of their application programming interfaces and formal models of microprocessor instruction set architectures.

These tools allow one to eliminate some sources of deviations of software implementation from its specification and the implied safety problems, however, they and their underlying theoretical frameworks focus on verification of relatively simple systems, primarily on systems of the following types:

- software which performs basic logical operations or numerical computations (e.g. software controllers);

- communication protocols which involve simple types of data.

Therefore such tools lack sufficiently easily usable methods of verification of

- software which performs complex processing of partially structured (semistructured) data;

- communication protocols which involve complex types of data.

Besides, their application is restricted because it requires specialized knowledge, is labour intensive, and time consuming.

These restrictions are a factor that may prevent expansion of the mentioned tools and theoretical frameworks outside of traditional safety-critical systems domains. Emerging high tech areas like the Internet of Things (IoT) rely on the idea of combining software systems and hardware devices and physical objects which involve complex interaction protocols, large-scale interaction, and processing of large volumes of semistructured data e.g. in home automation, smart buil-

dings, smart cities, etc. Errors in IoT software can impact the real world and lead to cyber security breaches or direct hazards to humans, but due to the nature of IoT systems in each particular case the potential impact of software errors is difficult to asses. Moreover, such errors are difficult to eliminate through testing, because IoT systems have to be able to function under variety of circumstances which are costly to model or reproduce. Thus IoT and other relevant high-tech areas could benefit from introduction of formal software verification approaches to their systems development processes.

Software for the Internet of Things (IoT) and other emerging high-tech areas has some differences from traditional safety-critical software which make application of the state-of-the-art verification tools to it rather difficult. One of them is processing of complex, usually semistructured types of data, instead of simple types of data such as logical values or numbers. Usually such data are encoded in data formats like JSON and XML which have tree-like, hierarchical nature.

2 General considerations about implementation of the approach

The work described in this paper aims to implement a formal verification tool which may overcome some limitations of the existing verification tools which prevent them to deal with software which processes complex, semistructured types of data.

The implementation of this tool is based on the composition-nominative approach to program formalization [1] [2] [3], and the Mizar system [4] [5], a software for formalizing mathematical theories (proof assistant) based on first-order logic and Tarski-Grothendieck set theory and a library of already formalized theories (Mizar Mathematical Library).

Composition-nominative approach provides the means of formalization of data – the notion of nominative data which is able to uniformly represent common forms of data used in programming (e.g. lists, trees, tables, multidimensional arrays, etc.), mathematical models of software

which operates on such data based on generalization of Glushkov algorithmic algebras [6], and a logic for reasoning about properties and correctness of such software – a generalized Floyd-Hoare logic [7] [8] with partial pre- and post-conditions for programs which operate on nominative data [9] [10]. The Mizar system provides an environment where the notions, models and logics of the composition-nominative approach can be formalized and implemented. In more detail the plan of their implementation in Mizar is described in [11].

A benefit of usage of Mizar as such an environment is that the Mizar Mathematical Library includes a large amount of notions and facts about continuous mathematics which allow formalization of mathematical models of physical aspects of IoT systems. More details on the link between the composition-nominative approach and the mathematical systems theory can be found in [12] [13].

3 Mizar Formalization

We formalized the basic notions of composition-nominative approach in a series of Mizar papers entitled NOMIN_1.MIZ [14]–NOMIN_4.MIZ and PARTPR_1.MIZ [15]. More specifically, we formalized

- the notion of a nominative data with simple names and complex values called the nominative data of type TND_{SC} [6];
- the main operations on nominative data of type TND_{SC} – *naming*, *denaming*, *overlapping* [6]; together with their domain they form an algebra called the nominative data algebra;
- the notion of a partial predicate on an arbitrary nonempty set;
- the logical compositions of *negation*, *conjunction*, *disjunction* [6] on partial predicates defined in accordance with the truth tables of Kleene's strong logic of indeterminacy [16]; together with the set of partial predicates they form an algebra which belongs to the class of Kleene algebras [17];

- the notion of a binominative function [6] which is a partial function mapping nominative data to nominative data; such functions serve as semantic models of sequential programs which process nominative data;

- the main compositions on binominative functions and partial predicates on nominative data [6] – *sequential composition* (of two binominative functions), *superposition* (of binominative functions into a binominative function or a predicate on nominative data), *assignment* (of the result of a binominative function to a name), *branching* ("if" operator), *loop* ("while" operator); together with the sets of binominative functions and predicates on nominative data and several chosen constants (null-ary compositions) they form an algebra called the nominative algorithmic algebra [18] which generalizes Glushkov algorithmic algebras [19];

- the notion of a Hoare triple [9], [7], [8] consisting of a precondition, a program, and a postcondition; the precondition and postcondition are partial predicates on nominative data; the program is a binominative function;

- the Floyd-Hoare composition [9];

- the rules of the extended Floyd-Hoare logic [9] for programs over nominative data (on semantic level) which allow reasoning about Hoare triples with partial pre- and post-conditions.

The mentioned elements allow one to

- represent semantics of sequential programs which process hierarchically organized data (modeled as nominative data) in the form of binominative functions in Mizar;

- formulate program properties in the form of Hoare triples with pre- and postconditions represented by partial predicates on nominative data;

- prove program properties using the rules of the extended Floyd-Hoare logic in Mizar.

The main elements of the mentioned Mizar formalization are described below. The details on syntax and semantics of the Mizar language can be found in [20].

The set of nominative data with simple names and complex values [6] over given (arbitrary, but fixed) nonempty sets of names V and atomic values A is the set

$$ND(V,A) = \bigcup_{k=0}^{\infty} ND_k(V,A),$$

where

$$ND_0(V,A) = A \cup \{\emptyset\},$$

$$ND_{k+1}(V,A) = A \cup \left(V \xrightarrow{n} ND_k(V,A)\right), \quad k \in \{0,1,2,...\}.$$

Here $V \xrightarrow{n} X$ denotes the set of all partial functions with finite graph from V to a set X.

The set $ND(V,A)$ and the associated notions are formalized in Mizar as follows (the detailed definitions can be found in [14] [21]):

- The Mizar mode `NonatomicND of V,A` represents the type of nonatomic nominative data, i.e. the type of elements of the set $ND(V,A)\backslash A$. Here V, A are set-valued parameters of the type.
- The mode `TypeSCNominativeData of V,A` represents the type of elements of the set $ND(V,A)$ (i.e. of either atomic or nonatomic nominative data). It is defined as

```
definition
 let V,A be set;
 mode TypeSCNominativeData of V,A -> set means
 it in A or it is NonatomicND of V,A;
end;
```

- The functor `ND(V,A) -> set` represents the set $ND(V,A)$ mentioned above. It is defined using `TypeSCNominativeData` as

```
definition
 let V,A be set;
 func ND(V,A) -> set equals
 the set of all D
  where D is TypeSCNominativeData of V,A;
end;
```

The main operations on nominative data are formalized in two ways (the detailed definitions can be found in [14]).

Firstly, each operation is defined as a functor which receives a nominative data as an input and returns a nominative data.

- The functor `naming(V,A,v,D) -> NonatomicND of V,A` (where `D` is of type `TypeSCNominativeData of V,A`) represents the naming operation $\Rightarrow v$ on $ND(V, A)$; here `V,A,v` are the operation's parameters (assumed to be fixed) and `D` is the operation's input. The operation's result is a nonatomic nominative data.

- The functor `denaming(v,D) -> TypeSCNominativeData of V,A` (where `D` is of type `NonatomicND of V,A`) represents the denaming operation $v \Rightarrow$ on $ND(V, A)$; here `V,A,v` are the operation's parameters (assumed to be fixed) and `D` is the operation's input. The operation's result is a nominative data.

- The functor `global_overlapping(V,A,d1,d2) -> TypeSCNominativeData of V,A` (where `d1,d2` are data of type `TypeSCNominativeData of V,A`) represents the binary overlapping operation ∇ on $ND(V, A)$. Here `V,A` are the operation's parameters (assumed to be fixed) and `d1,d2` are the operation's left and right input respectively.

Secondly, each operation is defined using a functor which returns the corresponding partial function on nominative data:

```
definition
 let V,A be set; let v be object;
 func naming(V,A,v)
   -> Function of ND(V,A),ND(V,A) means
```

```
 for D being TypeSCNominativeData of V,A holds
  it.D = naming(V,A,v,D);
end;

definition
 let V,A be set; let v be object;
 func denaming(V,A,v)
   -> PartFunc of ND(V,A),ND(V,A) means
 dom it = ND(V,A) \ A &
 for D being NonatomicND of V,A st not D in A
  holds it.D = denaming(v,D);
end;

definition
 let V,A be set; let v be object;
 func local_overlapping(V,A,v)
   -> PartFunc of [:ND(V,A),ND(V,A):],ND(V,A)
  means
 dom it = [: ND(V,A) \ A , ND(V,A) \ A :] &
 for d1,d2 being NonatomicND of V,A
  st not d1 in A & not d2 in A holds
 it. [d1,d2] = local_overlapping(V,A,d1,d2,v);
end;
```

Depending on the way of usage, one of these two formalizations of operations may be more convenient than another one.

The types of partial predicates on nominative data and binominative functions are defined as follows [18]:

```
definition
  let V,A;
  mode SCPartialNominativePredicate of V,A
     is PartFunc of ND(V,A),BOOLEAN;
  mode SCBinominativeFunction of V,A
     is PartFunc of ND(V,A),ND(V,A);
end;
```

Also, the sets of partial predicates on nominative data (`Pr(V,A)`) and binominative functions (`FPrg(V,A)`) are defined as follows:

```
definition
  let V,A;
  func Pr(V,A) -> set equals
  PFuncs(ND(V,A),BOOLEAN);
  coherence;
  func FPrg(V,A) -> set equals
  PFuncs(ND(V,A),ND(V,A));
  coherence;
end;
```

Binominative functions represent semantics of sequential programs which process nominative data (i.e. receive input data, perform certain computations and produce output data) and partial predicates represent semantics of conditions on nominative data.

Binominative functions corresponding to programs are constructed from the basic functions using operations (compositions) corresponding to programming language constructs like sequential execution, branching, loop.

These compositions were defined mathematically in [6]. The most important of them are formalized in Mizar as follows (the detailed formal definitions can be found in [18]).

- The functor

  `SCassignment(V,A,v) -> Function of FPrg(V,A),FPrg(V,A)`

  formalizes the unary assignment composition Asg^v which acts on the set of binominative functions $ND(V,A)\tilde{\to}ND(V,A)$ on data of type TND_{SC}. Here `v` is a parameter (name) to which a value is assigned. The result of application of the assignment composition $Asg^v(f)$ (where $f : ND(V,A)\tilde{\to}ND(V,A)$ is given) represents the semantics of the program consisting of the assignment statement $v := f$, where f is the semantics of the expression in program variables on the right hand side of the assignment statement.

- The functor `SCIF(V,A) ->`
  `Function of [:Pr(V,A),FPrg(V,A),FPrg(V,A):],FPrg(V,A)`
  formalizes the ternary branching composition IF. The result

of application of branching composition $IF(p, f, g)$, where $p : ND(V, A) \tilde{\to} \{T, F\}$ is a partial predicate, T, F are logical values ($True$, $False$), and $f, g : ND(V, A) \tilde{\to} ND(V, A)$ are binominative functions, represents semantics of the program consisting of the statement "**if** p **then** f **else** g".

- The functor `SCwhile(V,A) -> Function of [:Pr(V,A),FPrg(V,A):],FPrg(V,A)` formalizes the binary While loop composition WH. The result of application of the While loop composition $WH(p, f)$, where $p : ND(V, A) \tilde{\to} \{T, F\}$ is a partial predicate, T, F are logical values ($True$, $False$), and $f : ND(V, A) \tilde{\to} ND(V, A)$ is a binominative function, represents semantics of the program consisting of the statement "**while** p **do** f **end**".

The necessary compositions of partial predicates on arbitrary sets are formalized in PARTPR_1.MIZ [15].

Generally, the set of partial predicates (on an arbitrary nonempty set D) is formalized as follows:

```
definition
  let D;
  func Pr(D) -> set equals
  PFuncs(D,BOOLEAN);
  coherence;
end;
```

The compositions are formalized as follows:

- The functor `PPnegation(D) -> Function of Pr(D),Pr(D)` represents the negation on partial predicates on a set D.
- The functor `PPdisjunction(D) -> Function of [:Pr(D),Pr(D):],Pr(D)` represents the disjunction on partial predicates on a set D.

Alternatively, the negation and disjunction operations on predicates are formalized as functors `PP_not` and `PP_or` as follows:

```
definition
  let D,p,q;
  func PP_or(p,q) -> PartialPredicate of D equals
  PPdisjunction(D).(p,q);
  coherence;
  commutativity;
  idempotence;
end;

definition
  let D,p;
  func PP_not(p) -> PartialPredicate of D equals
  PPnegation(D).p;
  coherence;
  involutiveness;
end;
```

There are also specializations of these notions for predicates on nominative data.

- The functor

  `SC_not(p) -> SCPartialNominativePredicate of V,A`

  where `p` is of type `SCPartialNominativePredicate of V,A`, represents the negation on partial predicates on
  $$ND(V,A)\tilde{\to}\{T,F\}.$$

- The functor

  `SC_or(p,q) -> SCPartialNominativePredicate of V,A`

  where `p,q` are of type `SCPartialNominativePredicate of V,A`, represents the disjunction on partial predicates on
  $$ND(V,A)\tilde{\to}\{T,F\}.$$

Conjunction and implication are formalized as a derived compositions using negation and disjunction:

```
definition
  let V,A,p,q;
```

```
  func SC_and(p,q) -> SCPartialNominativePredicate of V,A equals
  SC_not SC_or(SC_not(p),SC_not(q));
  coherence;
  commutativity;
  idempotence;
  func SC_imp(p,q) -> SCPartialNominativePredicate of V,A equals
  SC_or(SC_not(p),q);
  coherence;
end;
```

For reasoning about properties of programs (binominative functions) on nominative data an extended Floyd-Hoare logic [9] is used.

Semantically, it is based on the notion of a Hoare triple formalized in Mizar as follows.

A *semantic Floyd-Hoare triple* is a triple (p, f, q), where

- $p, q : ND(V, A) \tilde{\to} \{T, F\}$ are partial predicates on $ND(V, A)$ called the pre- and post-condition respectively;
- $f : ND(V, A) \tilde{\to} ND(V, A)$ is a binominative function (representing semantics of a program),

such that for each data $d \in ND(V, A)$, if $p(d)$ is defined and true, $f(d)$ is defined, and $q(f(d))$ is defined, then $q(d)$ is true.

This means that whenever the precondition (p) is satisfied ($p(d)$ is true) on the input data (d) of the program (f), and the program terminates on the input data ($f(d)$ is defined), and the postcondition (q) is defined on the program's output ($f(d)$), the postcondition is satisfied on the program's output (i.e. $q(f(d))$ is true).

The set of all semantic Floyd-Hoare triples (for pre-/postconditions and binominative functions over $ND(V, A)$) is formalized in Mizar as follows:

```
definition
  let V,A;
  func SemanticFloydHoareTriples(V,A) -> set equals
  { <*p,f,q*> where p,q is SCPartialNominativePredicate of V,A,
                    f is SCBinominativeFunction of V,A :
```

```
   for d holds d in dom p & p.d = TRUE & d in dom f &
  f.d in dom q implies
    q.(f.d) = TRUE };
  coherence;
end;

notation
  let V,A;
  synonym SFHTs(V,A) for SemanticFloydHoareTriples(V,A);
end;
```

The type of semantic Floyd-Hoare triples is introduced as follows:

```
definition
  let V,A;
  mode SemanticFloydHoareTriple of V,A
    is Element of SemanticFloydHoareTriples(V,A);
  mode SFHT of V,A is Element of SFHTs(V,A);
end;
```

Then semantic versions of the rules of the extended Floyd-Hoare logic for programs over nominative data [11] are formalized in the form of theorems which state that under certain conditions, the results of application of certain compositions to predicates and binominative functions form a semantic Floyd-Hoare triple.

For example, the composition rule R_IF described in [11] is formalized as follows:

```
theorem
  <*SC_and(r,p),f,q*> is SFHT of V,A &
  <*SC_and(SC_not(r),p),g,q*> is SFHT of V,A implies
  <*p,SC_IF(r,f,g),q*> is SFHT of V,A
```

This theorem formalizes the fact that $(p, IF(r, f, g), q)$ is a semantic Floyd-Hoare triple, if both $(r \wedge p, f, q)$ and $(\neg r \wedge p, g, q)$ are semantic Floyd-Hoare triples.

Informally, this means that in order to prove that from the assumption that on the input data of the program

if r **then** f **else** g

the precondition p holds, after execution of this program the postcondition q holds, it is sufficient to show that

- if $r \wedge p$ holds before execution of f (the first branch), then q holds after execution of f
- if $\neg r \wedge p$ holds before execution of g (the second branch), then q holds after execution of g.

Other rules of the extended Floyd-Hoare logic for programs on nominative data are formalized in a similar way.

4 Conclusions

We have described the ongoing work on implementation of the composition-nominative approach to program formalization in Mizar proof assistant. In particular, we described the way in which hierarchical data, sequential programs and conditions on hierarchically organized data are formalized and the way in which rules of an extended Floyd-Hoare logic for reasoning about properties of such programs are represented in Mizar. We plan to use the obtained results as a basis for development of a formal verification tool for software which processes complex forms of data.

References

[1] N. S. Nikitchenko, "A composition nominative approach to program semantics," Department of Information Technology, Technical University of Denmark, Tech. Rep. IT-TR 1998-020, 1998.

[2] N. Nikitchenko, "Abstract computability of non-deterministic programs over various data structures," in *Perspectives of System Informatics: 4th International Andrei Ershov Memorial Conference, PSI 2001*, ser. Lecture Notes in Computer Science, D. Bjorner, M. Broy, and A. Zamulin, Eds., vol. 2244. Springer,

Berlin, Heidelberg, 2001, pp. 468–481. [Online]. Available: https://doi.org/10.1007/3-540-45575-2_45

[3] M. Nikitchenko, "Composition-nominative aspects of address programming," *Cybern Syst Anal*, no. 45: 864, 2009. [Online]. Available: https://doi.org/10.1007/s10559-009-9159-4

[4] G. Bancerek, C. Byliński, A. Grabowski, A. Korniłowicz, R. Matuszewski, A. Naumowicz, K. Pak, and J. Urban, "Mizar: State-of-the-art and beyond," in *Intelligent Computer Mathematics. CICM 2015*, ser. Lecture Notes in Computer Science, M. Kerber, J. Carette, C. Kaliszyk, F. Rabe, and V. Sorge, Eds., vol. 9150. Springer, Cham, 2015, pp. 261–279. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-20615-8_17

[5] A. Grabowski, A. Korniłowicz, and A. Naumowicz, "Four decades of Mizar," *Journal of Automated Reasoning*, vol. 55, no. 3, pp. 191–198, October 2015. [Online]. Available: http://dx.doi.org/10.1007/s10817-015-9345-1

[6] V. G. Skobelev, M. Nikitchenko, and I. Ivanov, "On algebraic properties of nominative data and functions," in *Information and Communication Technologies in Education, Research, and Industrial Applications: 10th International Conference, ICTERI 2014, Kherson, Ukraine, June 9–12, 2014, Revised Selected Papers*, V. Ermolayev, H. C. Mayr, M. Nikitchenko, A. Spivakovsky, and G. Zholtkevych, Eds. Cham: Springer International Publishing, 2014, pp. 117–138. [Online]. Available: https://doi.org/10.1007/978-3-319-13206-8_6

[7] R. Floyd, "Assigning meanings to programs," *Mathematical aspects of computer science*, vol. 19, no. 19–32, 1967.

[8] C. Hoare, "An axiomatic basis for computer programming," *Commun. ACM*, vol. 12, no. 10, pp. 576–580, 1969.

[9] A. Kryvolap, M. Nikitchenko, and W. Schreiner, "Extending Floyd-Hoare logic for partial pre- and postconditions," in *Information and Communication Technologies in Education, Research, and Industrial Applications: 9th International Conference, ICTERI 2013, Kherson, Ukraine, June 19–22, 2013, Revised Selected Papers*, V. Ermolayev, H. C. Mayr, M. Nikitchenko, A. Spivakovsky, and G. Zholtkevych, Eds. Cham: Springer International Publishing, 2013, pp. 355–378. [Online]. Available: https://doi.org/10.1007/978-3-319-03998-5_18

[10] M. Nikitchenko and A. Kryvolap, "Properties of inference systems for Floyd-Hoare logic with partial predicates," *Acta Electrotechnica et Informatica*, vol. 13, no. 4, pp. 70–78, 2013. [Online]. Available: https://doi.org/10.15546/aeei-2013-0052

[11] A. Korniłowicz, A. Kryvolap, M. Nikitchenko, and I. Ivanov, "An approach to formalization of an extension of Floyd-Hoare logic," in *Proceedings of the 13th International Conference on ICT in Education, Research and Industrial Applications. Integration, Harmonization and Knowledge Transfer, Kyiv, Ukraine, May 15–18, 2017*, ser. CEUR Workshop Proceedings, V. Ermolayev, N. Bassiliades, H.-G. Fill, V. Yakovyna, H. C. Mayr, V. Kharchenko, V. Peschanenko, M. Shyshkina, M. Nikitchenko, and A. Spivakovsky, Eds., vol. 1844. CEUR-WS.org, 2017, pp. 504–523. [Online]. Available: http://ceur-ws.org/Vol-1844/10000504.pdf

[12] I. Ivanov, "On representations of abstract systems with partial inputs and outputs," in *Theory and Applications of Models of Computation: 11th Annual Conference, TAMC 2014, Chennai, India, April 11–13, 2014. Proceedings*, T. V. Gopal, M. Agrawal, A. Li, and S. B. Cooper, Eds. Cham: Springer International Publishing, 2014, pp. 104–123. [Online]. Available: https://doi.org/10.1007/978-3-319-06089-7_8

[13] I. Ivanov, "On local characterization of global timed bisimulation for abstract continuous-time systems," in *Coalgebraic Methods*

in Computer Science: 13th IFIP WG 1.3 International Workshop, CMCS 2016, Colocated with ETAPS 2016, Eindhoven, The Netherlands, April 2–3, 2016, Revised Selected Papers, I. Hasuo, Ed. Cham: Springer International Publishing, 2016, pp. 216–234. [Online]. Available: https://doi.org/10.1007/978-3-319-40370-0_13

[14] I. Ivanov, M. Nikitchenko, A. Kryvolap, and A. Korniłowicz, "Simple-named complex-valued nominative data – definition and basic operations," *Formalized Mathematics*, vol. 25, no. **3**, pp. 205–216, 2017. [Online]. Available: http://dx.doi.org/10.1515/forma-2017-0020

[15] A. Korniłowicz, I. Ivanov, M. Nikitchenko, and A. Kryvolap, "Kleene algebra of partial predicates," *Submitted to Formalized Mathematics*, 2018.

[16] S. Kleene, *Introduction to metamathematics*. North-Holland Publishing Co., Amsterdam, and P. Noordhoff, Groningen, 1952.

[17] D. Brignole and A. Monteiro, *Caractérisation des algèbres de Nelson par des egalités*. Bahía Blanca [Argentina]: Instituto de Matemática, Universidad Nacional del Sur, 1964.

[18] A. Korniłowicz, A. Kryvolap, M. Nikitchenko, and I. Ivanov, "Formalization of the nominative algorithmic algebra in Mizar," in *Information Systems Architecture and Technology: Proceedings of 38th International Conference on Information Systems Architecture and Technology – ISAT 2017: Part II*, J. Świątek, L. Borzemski, and Z. Wilimowska, Eds. Cham: Springer International Publishing, 2018, pp. 176–186. [Online]. Available: https://doi.org/10.1007/978-3-319-67229-8_16

[19] V. Glushkov, "Automata theory and formal transformations of microprograms," *Cybernetics (in Russian)*, vol. 5, pp. 3–10, 1965.

[20] F. Wiedijk. Writing a Mizar article in nine easy steps. https://www.cs.ru.nl/~freek/mizar/mizman.pdf.

[21] A. Korniłowicz, A. Kryvolap, M. Nikitchenko, and I. Ivanov, "Formalization of the algebra of nominative data in Mizar," in *Proceedings of the 2017 Federated Conference on Computer Science and Information Systems, FedCSIS 2017, Prague, Czech Republic, September 3–6, 2017.*, M. Ganzha, L. A. Maciaszek, and M. Paprzycki, Eds., 2017, pp. 237–244. [Online]. Available: https://doi.org/10.15439/2017F301

Ievgen Ivanov[1], Artur Korniłowicz[2], Mykola Nikitchenko[3]

Received December 30, 2017

[1] Taras Shevchenko National University of Kyiv, Ukraine
E–mail: ivanov.eugen@gmail.com

[2] University of Białystok, Poland
E–mail: arturk@math.uwb.edu.pl

[3] Taras Shevchenko National University of Kyiv, Ukraine
E–mail: nikitchenko@unicyb.kiev.ua