Satisfiability Problems in Quasiary Program Logics

Mykola Nikitchenko, Stepan Shkilniak, Valentyn Tymofieiev

Department of Theory and Technology of Programming, Taras Shevchenko National University of Kyiv, UKRAINE,
60 Volodymyrska Street, City of Kyiv, Ukraine, 01033, email: mykola.nikitchenko@gmail.com

Abstract: **In the paper we present special program specification algebras and logics defined for classes of quasiary mappings. Informally speaking, such mappings are partial mappings defined over partial states (partial assignments) of variables. Conventional *n*-ary mappings can be considered as a special case of quasiary mappings. Such mappings better reflect properties of software systems. We describe methods of reducibility of the satisfiability problem in quasiary logics to the satisfiability problem in logics of *n*-ary mappings. The methods proposed can be useful for software verification.**

Keywords: ***n*-ary mapping, quasiary mapping, program algebra, specification logic.**

I. Introduction

Algebraic approach to software system specification has the following two characteristics: 1) the formalism of many-sorted algebras is used to model such systems; 2) special logics based on such many-sorted algebras are used to reason about system properties. In the literature various kinds of such algebras and logics are described (e.g. see [1, 2]).

In this paper we present special algebras and logics defined for classes of quasiary mappings. Informally speaking, such mappings are partial mappings defined over partial states (partial assignments) of variables. Conventional *n*-ary mappings can be considered as a special case of quasiary mappings. Quasiary mappings better reflect properties of software systems therefore construction and investigation of algebras and logics of quasiary mapping is an important challenge.

Proposed constructions are based on a composition-nominative approach [3]. Principles of the approach (*development* of program notions *from abstract to concrete*, *priority of semantics*, *compositionality* of programs, and *nominativity* of program data) specify program models as *composition-nominative systems* which consist of *composition*, *description*, and *denotation* systems. A *composition* system defines semantic aspects of programs, a *description* system defines program descriptions (syntactic aspects), and a *denotation* system specifies meanings (referents) of descriptions. We consider semantics of programs as partial functions over a class of data processed by programs; compositions are *n*-ary operations over functions. Thus, a composition system can be specified by two algebras: *data algebra* and *function algebra*. Function algebra is the main *semantic notion* in program formalization. *Terms* of this algebra define *syntax* of programs (description system), and ordinary procedure of term interpretation gives a *denotation* system.

The constructed program models form a base for developing special program logics called *composition-nominative logics* (CNL).

In this paper we continue our work on studying CNL [4–6] focusing on quasiary specification algebras and logics. The main questions under discussion concern satisfiability problems and their reduction to satisfiability problems in logics of *n*-ary mappings.

II. Quasiary Mappings

Quasiary mappings can be met in different branches of mathematics, logics, and computer science. Informally speaking, such mappings appear when we use variables (names) to construct mapping arguments. Here we consider only usage of quasiary mappings in logic semantics and formal models of programs.

The notion of quasiary predicate and function can be easily understood when we analyze Tarski's definition of first-order language semantics. This semantics is based on the notion of interpretation which consists of two parts: 1) interpretation of predicate and function symbols in some structure, and 2) interpretation of individual variables in the domain of this structure. The latter are usually called *variable assignments (or valuations)* and can be represented by total mappings from a set of individual variables (names) V into some set of basic values A. The class of such total mappings will be denoted $V \xrightarrow{t} A$ or A^V, and called *total nominative sets*. Thus, Tarski's semantics interprets predicate and function symbols as total quasiary predicates and functions defined on the class A^V of total nominative sets. In applications like model checking, program verification, automated theorem proving, etc., partial assignments (nominative sets) are often used instead of total assignments. The class of such partial mappings will be denoted $V \xrightarrow{p} A$ or ${}^VA$, and called *partial nominative sets* (partial data); the term 'partial is often omitted. Predicates and functions over nominative sets are called *quasiary*. This means that formulas and terms can be interpreted as quasiary predicates and functions respectively.

Quasiary mappings also appear in a natural way in denotational semantics of programs. In this semantics program states are represented as nominative sets, Boolean expressions as *quasiary predicates* of the type $Pr_A^V = {}^VA \xrightarrow{p} Bool$, arithmetical expressions as *quasiary functions* of the type $Fn_A^V = {}^VA \xrightarrow{p} A$, and program statements as *bi-quasiary functions* (*program functions*) of the type $PF_A^V = {}^VA \xrightarrow{p} {}^VA$. Semantics of structured statements is defined by the following compositions with conventional meaning: *assignment composition* AS^x (x is a parameter from V), *composition of sequential execution* $\bullet$, *conditional*

composition IF, *loop composition WH*. For structural expressions we additionally use unary denomination composition $'x$ and various superpositions.

Thus, we obtain a program algebra with three carriers: quasiary predicates, quasiary functions, and bi-quasiary functions (program functions). Such algebras can be called *algorithmic algebras*.

To extend such algebras to program specification algebras we add *quantifiers* and *prediction composition* '$\cdot$'. Prediction composition is simply a functional composition of a program function and a predicate. This composition is strong enough to represent Hoare assertions, and therefore, specification algebras with these compositions are rather expressive [7].

In the rest of the paper we consider *program specification algebras* and *logics of partial quasiary mappings.*

To emphasize a mapping's *partiality/totality* we write the sign $\xrightarrow{p}$ for partial mappings and the sign $\xrightarrow{t}$ for total mappings. Given a partial mapping $\mu, \mu' : D \xrightarrow{p} D'$, d, $d' \in D$ we write:
– $\mu(d)\downarrow$ ($\mu(d)\uparrow$) to denote that μ is defined (undefined) on d;
– $\mu(d)\downarrow = d'$ to denote that μ is defined on d with a value d';
– $\mu(d) \cong \mu'(d')$ to denote the strong equality.

We omit proofs and some details of complicated definitions.

III. Quasiary Specification Algebras

We use the following set of composition symbols parameterized by V:

$$Cs^V = \{\vee, \neg, \exists x, 'x, S_P^{v_1,...,v_n}, S_F^{v_1,...,v_n}, =, S_G^{v_1,...,v_n}, AS^x, \bullet, IF, WH, \cdot\}$$

Formal definitions of compositions can be found in [4, 7, 8]. Compositions $S_P^{v_1,...,v_n}$, $S_F^{v_1,...,v_n}$, and $S_G^{v_1,...,v_n}$ are called superpositions and represent substitutions of quasiary functions into a predicate, function, and program function respectively. We also denote such compositions as $S_P^{\bar{v}}, S_F^{\bar{v}}$, $S_G^{\bar{v}}$.

A tuple $\mathfrak{A}_A^V(Cs^V) =< Pr_A^V, Fn_A^V, PF_A^V; Cs^V >$ is called *quasiary specification algebra* (QSA) over V and A.

Variables used as composition parameters can be classified as *essential* in the sense that they can affect the result of composition application evaluation and as *updatable* in the sense that the values of these variables can change during evaluation. Variable x is essential for denomination composition $'x$, variable x is updatable for $\exists x$ and AS^x, variables $v_1,...,v_n$ are updatable for $S_P^{v_1,...,v_n}$, $S_F^{v_1,...,v_n}$, and $S_G^{v_1,...,v_n}$.

Now we describe the main properties of superpositions.

Lemma 1 (superposition folding). Let $\bar{y} = y_1,...,y_n$, $\overline{f'} = f_1',...,f_n'$; $\bar{x} = x_1,...,x_k$, $\bar{f} = f_1,...,f_k$, $\overline{h'} = h_1',...,h_k'$, $\bar{v} = v_1,...,v_m$, $\bar{h} = h_1,...,h_m$, $\{y_1,...,y_n\} \cap \{v_1,...,v_m\} = \varnothing$. Then the following properties hold in $\mathfrak{A}_A^V(Cs^V)$:

SS$_P$. $S_P^{\bar{y},\bar{x}}(S_P^{\bar{x},\bar{v}}(p, \bar{f}, \bar{h}), \overline{f'}, \overline{h'}) = S_P^{\bar{y},\bar{x},\bar{v}}(p, \bar{\sigma})$,

SS$_F$. $S_F^{\bar{y},\bar{x}}(S_F^{\bar{x},\bar{v}}(f, \bar{f}, \bar{h}), \overline{f'}, \overline{h'}) = S_F^{\bar{y},\bar{x},\bar{v}}(f, \bar{\sigma})$,

SS$_G$. $S_G^{\bar{y},\bar{x}}(S_G^{\bar{x},\bar{v}}(g, \bar{f}, \bar{h}), \overline{f'}, \overline{h'}) = S_G^{\bar{y},\bar{x},\bar{v}}(g, \bar{\sigma}))$,

where

$$\bar{\sigma} = (\overline{f'}, S_F^{\bar{y},\bar{x}}(f_1, \overline{f'}, \overline{h'}),..., S_F^{\bar{y},\bar{x}}(f_k, \overline{f'}, \overline{h'}),$$
$$S_F^{\bar{y},\bar{x}}(h_1, \overline{f'}, \overline{h'}),..., S_F^{\bar{y},\bar{x}}(h_m, \overline{f'}, \overline{h'})).$$

Lemma 2 (distributivity of superposition). The following properties hold in $\mathfrak{A}_A^V(Cs^V)$:

S$\vee$. $S_P^{\bar{v}}(p \vee q, \bar{f}) = S_P^{\bar{v}}(p, \bar{f}) \vee S_P^{\bar{v}}(q, \bar{f})$,

S$\neg$. $S_P^{\bar{v}}(\neg p, \bar{f}) = \neg S_P^{\bar{v}}(p, \bar{f})$,

S=. $S_P^{\bar{v}}(h_1 = h_2, \bar{f}) = (S_F^{\bar{v}}(h_1, \bar{f}) = S_F^{\bar{v}}(h_2, \bar{f}))$.

S$\exists$. $S_P^{\bar{v}}(\exists xp, \bar{f}) = \exists u\, S_P^{\bar{v}}(S_P^x(p, 'u), \bar{f})$, $u \neq x$, $u \notin \bar{v}$, u is unessential for p and $\bar{f}$,

(here "u is unessential for p and $\bar{f}$" means that $p(d) \cong p(d')$ and $\bar{f}(d) \cong \bar{f}(d')$ for any d and d' such that $d \|_{-\{u\}} = d' \|_{-\{u\}}$).

Superposition compositions are not distributive with respect to *WH*, therefore we simplify superpositions into program functions using the identity program function *id*.

Lemma 3 (superpositions with program functions). The following properties hold in $\mathfrak{A}_A^V(Cs^V)$:

SG. $S_G^{\bar{v}}(g, \bar{f}) = S_G^{\bar{v}}(id, \bar{f}) \bullet g$ – superposition into program function,

SP. $S_P^{\bar{v}}(g \cdot p, \bar{f}) = S_G^{\bar{v}}(g, \bar{f}) \cdot p$ – superposition with prediction composition.

Lemma 4 (superposition simplification). The following properties hold in $\mathfrak{A}_A^V(Cs^V)$:

SE. $S_P(p) = p$, $S_F(f) = f$, $S_G(g) = g$ – superpositions with empty parameter list,

SiD. $S_F^{\bar{v}}('x, \bar{f}) = 'x$ if $x \notin \bar{v}$, $S_F^{x,\bar{v}}('x, f, \bar{h}) = f$ – superpositions into denomination functions,

SwD. $S_P^{x,\bar{v}}(p, 'x, \bar{f}) = S_P^{\bar{v}}(p, \bar{f})$, $S_F^{x,\bar{v}}(h, 'x, \bar{f}) = S_F^{\bar{v}}(h, \bar{f})$ – superpositions with denomination function,

ST. $S_P^{\bar{v},x,y,\bar{z}}(\varphi, \bar{f}, h, h', \overline{f'}) = S_P^{\bar{v},y,x,\bar{z}}(\varphi, \bar{f}, h', h, \overline{f'})$,
$S_F^{\bar{v},x,y,\bar{z}}(f, \bar{f}, h, h', \overline{f'}) = S_F^{\bar{v},y,x,\bar{z}}(f, \bar{f}, h', h, \overline{f'})$,
$S_G^{\bar{v},x,y,\bar{z}}(g, \bar{f}, h, h', \overline{f'}) = S_G^{\bar{v},y,x,\bar{z}}(g, \bar{f}, h', h, \overline{f'})$ – transposition of parameters.

These properties will be used to construct superpositional normal forms for language expressions.

Now we study relations between QSA $\mathfrak{A}_A^V(Cs^V)$ and QSA $\mathfrak{A}_A^{V'}(Cs^{V'})$ induced by the following two relations between sets of their names:

1) there is a renomination bijection $\beta : V \xrightarrow{t} V'$,

2) V' is an extension of V ($V \subseteq V'$).

In the first case β induces in a natural way new mappings $\beta_P : Pr_A^V \xrightarrow{t} Pr_A^{V'}$, $\beta_F : Fn_A^V \xrightarrow{t} Fn_A^{V'}$, and $\beta_G : PF_A^V \xrightarrow{t} PF_A^{V'}$ with the following properties.

Theorem 1. Mappings β_C, β_P, β_F, and β_G define an isomorphism of QSA $\mathfrak{A}_A^V(Cs^V)$ and QSA $\mathfrak{A}_A^{V'}(Cs^{V'})$.

Theorem 2. Let $V \subseteq V'$. Then inclusion mapping induces (ignoring variables from $U = V' \setminus V$) an injective homomorphism of $\mathfrak{A}_A^V(Cs^V)$ into $\mathfrak{A}_A^{V'}(Cs^V)$.

Now we can study an algebra with mappings over total data that "mimic" mappings over partial data. A special element ε (ε∉*A*) will represent a case when a value of a variable or a function is undefined. Let $A_\varepsilon = A \cup \{\varepsilon\}$ and $A_\varepsilon{}^V = V \xrightarrow{t} A \cup \{\varepsilon\}$. We construct a QSA $\mathfrak{A}_{A,\varepsilon}^V(Cs_\varepsilon^V)$ with total data that "mimics" QSA $\mathfrak{A}_A^V(Cs^V)$. Carriers of the new algebra are classes $Pr_{A,\varepsilon}^V = A_\varepsilon^V \xrightarrow{p} Bool$, $FnT_{A,\varepsilon}^V = A_\varepsilon^V \xrightarrow{t} A_\varepsilon$, and $PF_{A,\varepsilon}^V = A_\varepsilon^V \xrightarrow{p} A_\varepsilon^V$.

Then we define mapping $\varepsilon_D^+ : {}^V A \xrightarrow{t} A_\varepsilon^V$ that "add ε into a nominative set. This mapping induces mappings ε_P^+, ε_F^+, and ε_G^+ relating corresponding carriers.

Theorem 3. Mappings ε_P^+, ε_F^+, and ε_G^+ define an isomorphism of $\mathfrak{A}_A^V(Cs^V)$ and $\mathfrak{A}_{A,\varepsilon}^V(Cs_\varepsilon^V)$.

We treat *n*-ary operations as a special case of quasiary mappings with the set of variables $N=\{1,\ldots, n\}$ and total data. In this case a total nominative set $[1 \mapsto a_1, \ldots, n \mapsto a_n]$ is represented by a tuple $(a_1, \ldots, a_n)$. Thus, all algebra mappings are defined on a Cartesian product A^n. Compositions from Cs^N can be treated as compositions over *n*-ary mappings.

Here we do not redefine compositions in this style assuming that it is a simple task. The term 'unified' means that all mappings have the same arity.

Obtained algebra is called a unified *n*-ary specification algebra (NSA) and is denoted $\mathfrak{A}_A^N(Cs^N)$. The following proposition is practically an immediate consequence of Theorems 1–3.

Theorem 4. Let $V = \{v_1, \ldots, v_n\}$, $N = \{1, \ldots, n\}$ $(n \geq 1)$, $\beta : V \xrightarrow{t} N$ be a bijection, $\mathfrak{A}_A^V(Cs^V)$ be QSA and $\mathfrak{A}_{A,\varepsilon}^N(Cs^N)$ be NSA $(\varepsilon \notin A)$. Then mappings β_C, $\beta_P \circ \varepsilon_P^+$, $\beta_F \circ \varepsilon_F^+$, and $\beta_G \circ \varepsilon_G^+$ define an isomorphism of QSA $\mathfrak{A}_A^V(Cs^V)$ and NSA $\mathfrak{A}_{A,\varepsilon}^N(Cs^N)$.

IV. Quasiary Specification Logic

To define a *quasiary specification logic*, denoted $\mathfrak{L}^Q$, we have to specify its semantic, syntactic, and interpretational components [4, 8].

Semantic components of $\mathfrak{L}^Q$ is based on the class of quasiary specification algebras $\mathfrak{A}_A^V(Cs^V)$ for different *A*.

A syntactic component specifies the language of $\mathfrak{L}^Q$ constructed over signature $\Sigma_Q^V = (Cs^V, Ps, Fs, Pgs)$ where *Ps*, *Fs*, and *Pgs* are respectively the sets of predicate symbols, ordinary function symbols, and program function symbols. For simplicity, we use the same notation for symbols of compositions and compositions themselves.

For a given signature Σ_Q^V the set of formulas $Fr(\Sigma_Q^V)$, the set of terms $Tr(\Sigma_Q^V)$, and the set of programs $Pg(\Sigma_Q^V)$ are defined by induction in a traditional way.

Interpretational component is defined in the following way. Given $\Sigma_Q^V = (Cs^V, Ps, Fs, Pgs)$ and a set *A* we can define a QSA $\mathfrak{A}_A^V(Cs^V) = < Pr_A^V, Fn_A^V, PF_A^V; Cs^V >$. Composition symbols have fixed interpretation, but we additionally need interpretations $I^{Ps} : Ps \xrightarrow{t} Pr_A^V$, $I^{Fs} : Fs \xrightarrow{t} Fn_A^V$, and $I^{Pgs} : Pgs \xrightarrow{t} PF_A^V$ of predicate, function, and program function symbols respectively. A tuple $J = (\Sigma_Q^V, A, I^{Ps}, I^{Fs}, I^{Pgs})$ is called *an* $\mathfrak{L}^Q$*-interpretation*. Usually the prefix $\mathfrak{L}^Q$ is omitted. Given an interpretation *J* we denote meanings in *J* of a formula Φ, a term *t*, and a program π respectively Φ_J, t_J, and π_J.

$\mathfrak{L}^Q$-formula Φ is *satisfiable in an interpretation J* if there exists an element *d* such that $\Phi_J(d)\downarrow = T$. This is denoted $\mathfrak{L}^Q, J \models \Phi$. Formula Φ is *satisfiable in the logic* $\mathfrak{L}^Q$ ($\mathfrak{L}^Q \models \Phi$), if there exists an interpretation *J* such that $\mathfrak{L}^Q, J \models \Phi$. Formulas Φ and Ψ are *equisatisfiable*, if they are both satisfiable or both unsatisfiable.

$\mathfrak{L}^Q$-formula Φ is called *valid in an interpretation J* if there is no *d* such that $\Phi_J(d)\downarrow = F$. This is denoted $\mathfrak{L}^Q, J \models \Phi$, which means that Φ is not refutable in *J*. A formula Φ is called *valid* in $\mathfrak{L}^Q$ if $\mathfrak{L}^Q, J \models \Phi$ for any interpretation *J*. We shall denote this $\mathfrak{L}^Q \models \Phi$, or just $\models \Phi$ if the logic in hand is understood from the context.

$\mathfrak{L}^Q$-formulas Φ and Ψ are *equivalent*, if for every *J* predicates Φ_J and Ψ_J are identical. Such notion of equivalence can be also defined for terms and programs.

Validity and satisfiability problems for QSL are related in the following way: Φ is valid in *J* if and only if $\neg\Phi$ is unsatisfiable in *J*.

Let $N=\{1,\ldots,n\}$. We treat a unified *n*-ary specification logic $\mathfrak{L}^N$ as a quasiary specification logic with a signature $\Sigma_N^n = (Cs_n^{\{1,\ldots,n\}}, Ps, Fs, Pgs)$ constructed over total nominative sets. This logic is semantically based on unified *n*-ary specification algebras.

Now we will study a problem how to relate $\mathfrak{L}^Q$ and $\mathfrak{L}^N$ with respect to the satisfiability problem, namely, given $\mathfrak{L}^Q$-formula Φ construct $\mathfrak{L}^N$-formula Φ^n such that Φ and Φ^n will be equisatisfiable. We do this in several steps:

- introducing a logic $\mathfrak{L}^{QU}$ with unessential variables,
- constructing a superpositional normal form Φ^s of Φ in $\mathfrak{L}^{QU}$,
- introducing a logic $\mathfrak{L}^{QUR}$ with finitely restricted sets of updatable variables,
- constructing a unified superpositional normal form Φ^u of Φ^s in $\mathfrak{L}^{QUR}$,
- constructing from Φ^s a formula Φ^t of logic $\mathfrak{L}_T^{QUR}$ with total data,
- translating Φ^t into $\mathfrak{L}^N$-formula Φ^n,

– proving equisatisfiability of Φ and Φ^n.

Logic $\mathfrak{L}^Q$ being a rather powerful logic still is not expressible enough to represent various important transformations. Therefore we introduce as its extension *a logic with unessential variables* denoted $\mathfrak{L}^{QU}$. Here U is an infinite set of variables such that $V \cap U = \varnothing$. *Unessential variables* do not affect the meaning of formulas (terms, programs) [4,8]. An additional requirement is that unessential variables are not updatable by programs. The signature of $\mathfrak{L}^{QU}$ is $\Sigma_Q^{V,U} = (Cs^{V \cup U}, Ps, Fs, Pgs)$.

The following statement is a consequence of Theorem 2.

Lemma 5 ($\mathfrak{L}^{QU}$ is a model-theoretic conservative extension of $\mathfrak{L}^Q$). Let $\mathfrak{L}^{QU}$-interpretation J^U be an unessential extension of $\mathfrak{L}^Q$-interpretation J, $\Phi \in Fr(\Sigma_Q^V)$, $t \in Tr(\Sigma_Q^V)$, $\pi \in Pg(\Sigma_Q^V)$. Then

$$\iota_P^U(\Phi_J) = \Phi_{J^U},\ \iota_F^U(t_J) = t_{J^U}, \text{ and } \iota_G^U(\pi_J) = \pi_{J^U}.$$

Introduction of $\mathfrak{L}^{QU}$ permits to formulate transformations rules based on properties presented in Lemmas 1–4.

$\mathfrak{L}^{QU}$-formula Φ is said to be in *superpositional normal form*, if the following conditions hold:

SP. For each subformula $S_P^{\bar{v}}(\Psi, \bar{t})$ of formula Φ we have that $\Psi \in Ps$;

SF. For each subformula of the form $S_F^{\bar{v}}(t, \bar{t})$ we have that $t \in Fs$;

SG. For each subformula of the form $S_G^{\bar{v}}(\pi, \bar{t})$ we have that $\pi = id$.

Lemma 6. Let $\Phi \in Fr(\Sigma_Q^{V \cup U})$. Then, using transformation specified by Lemmas 1-4, a superpositional normal form Φ^s of Φ can be constructed such that $\Phi^s \approx \Phi$.

In a similar way we can define transformations that first lead to a formula Φ^t of logic $\mathfrak{L}_T^{QUR}$ with total data and then to formula Φ^n of logic $\mathfrak{L}^N$.

V. Reduction of the Satisfiability Problem

Combining all obtained results, we can prove the following main theorem that states reducibility of the satisfiability problem in quasiary specification logics with finitely restricted sets of updatable variables to the satisfiability problem in *n*-ary specification logics.

Theorem 5. Let Φ be a $\mathfrak{L}^{QUR}$-formula and Φ^n be a $\mathfrak{L}^N$-formula obtained by the above-described transformations. Then Φ and Φ^n are equisatisfiable.

Results of such kind permit to use existing satisfiability checkers for classical predicate and program logics, based on n-ary mappings, to check satisfiability of formulas for quasiary logics.

VI. Conclusion

In this paper, we have developed special program specification algebras and logics defined for classes of quasiary mappings. These algebras and logics reflect such features of software systems as partiality of data, partiality and unrestricted arity of predicate and functions, sensitivity to unassigned variables. For the constructed logics some laws of classical logic fail. We have studied relations of quasiary logics to logics of *n*-ary mapping. Obtained results demonstrate that logics of quasiary mappings are more powerful and expressive than logics based on *n*-ary mappings. We have developed methods of reduction of the satisfiability problems in quasiary logics to the satisfiability problems for logics based on *n*-ary mappings. Such methods can be useful for construction and investigation of logics for program reasoning.

Future work on the topic will include construction of calculi for important fragments of the considered logics. Also, a prototype of software systems for theorem proving in quasiary specification logics should be developed. First steps in this direction are made in [9, 10].

References

[1] Handbook of Logic in Computer Science, S. Abramsky, Dov M. Gabbay, and T. S. E. Maibaum (eds.), in 5 volumes, Oxford Univ. Press, Oxford, 1993–2001.

[2] D. Sannella, A. Tarlecki, “Foundations of Algebraic Specification and Formal Software Development”, Springer, 2012.

[3] N.(M.) Nikitchenko, “A Composition Nominative Approach to Program Semantics”. *Technical Report IT–TR 1998-020,* Technical University of Denmark, 103 p., 1998.

[4] M. Nikitchenko, V. Tymofieiev, “Satisfiability in Composition-Nominative Logics”, *Central European Journal of Computer Science*, vol. 2, issue 3, 2012, pp. 194-213.

[5] M. Nikitchenko, S. Shkilniak, “Applied Logic”, Publishing house of Taras Shevchenko National University of Kyiv, Kyiv, 2013 (in Ukrainian), 278 p.

[6] M. Nikitchenko, V. Tymofieiev, “Composition-Nominative Logics in Rigorous Development of Software Systems”, *LNBIP*, vol. 137, pp. 140–151. Springer, Heidelberg, 2013.

[7] A. Kryvolap, M. Nikitchenko, W. Schreiner, “Extending Floyd-Hoare logic for partial pre- and postconditions”, *CCIS*, vol. 412, pp. 355-378, Springer, Heidelberg, 2013.

[8] M. Nikitchenko, S. Shkilniak, “Algebras and logics of partial quasiary predicates”, *Algebra and Discrete Mathematics*, vol. 23, number 2, 2017, pp. 263–278.

[9] I. Ivanov, M. Nikitchenko, A. Kryvolap, A. Kornilowicz, “Simple-Named Complex-Valued Nominative Data – Definition and Basic Operations”, *Formalized Mathematics*, 25(3), pp. 205-216, 2017.

[10] A. Kornilowicz, A. Kryvolap, M. Nikitchenko, I. Ivanov, “Formalization of the nominative algorithmic algebra in Mizar”, *Advances in Intelligent Systems and Computing*, vol. 656, pp. 176–186, Springer, 2017.