ON COMBINATION OF DEDUCTION AND ANALYTICAL TRANSFORMATIONS IN E-LEARNING TESTING

Vitaly Klimenko, Alexander Lyaletski, Mykola Nikitchenko

Abstract: *We investigate a possible way for solving the problem of combination of logical inference search methods and symbolic computation tools in e-learning testing on the basis of the approaches developed at the Kiev schools of automated theorem proving and analytical transformations. The investigations started in the first half of 1960s at the Institute of Cybernetics of the Academy of Sciences of Ukraine. Some years later the Faculty of Cybernetics of the Kiev State University was involved in the corresponding projects. The current state of investigations on the topic as well as their theoretical and practical background is described in the paper.*

Keywords: *analytical transformation, automated theorem proving, deduction, e-learning, intelligent tutoring system.*

ACM Classification Keywords: *I.2.3 Deduction and Theorem Proving – Deduction. I.2.4 Knowledge Representation Formalisms and Methods – Predicate logic. G.4 Mathematical software. K.3.2 Computer and Information Science Education.*

Introduction

At the beginning of the 1960th, Academician V.M. Glushkov initiated two approaches to the development and implementation of computer-aided mathematics: one was concerned with symbolic computations (i.e. computer algebra systems in the modern terminology) and the other with automated theorem-proving (i.e. automated reasoning systems in more general sense). Now, these two approaches occupy an important place in information technologies and intelligent tutoring systems, in particular. That is why it is interesting to know what impact that their combination may have on the development of intelligent testing in e-learning in current days. In this connection, we first describe the approaches and discuss their impact after this.

Deduction in testing

Deductive testing consists in logical verification of reasoning steps expressed in a formal language. In accordance with Glushkov's paper [Glushkov, 1970], the language should be

formal and maximally be close to a natural language. Deductive testing is to be used in mathematical disciplines having the form of formal axiomatic theories containing logical inference rules. It also can be useful (within intelligent tutoring systems) for applying it in other applied domains, for example, in jurisprudence, where the testing consists in performing legally and logically valid reasoning steps, or in creating legal documents consistent with the current legislation.

Deductive approach

The deductive approach itself is based on the declarative way of representation and logical processing of knowledge having the form of formalized texts (containing axioms, definitions, propositions, and so on, when we deal with mathematical problems). Systems exploiting it usually are called automated reasoning systems or, in particular, systems for automated theorem proving. Note that this approach turns out to be the most adequate for the automated logical inference search as well as for verification of a formal text (mathematical or not), namely, checking the validity of all the reasoning steps in it.

For the purposes of deductive testing, we adhere to the following requirements for a testing environment:

- For presentation of reasoning, a trainee must use a (semi-)formal language which is close to the natural language of mathematical publications. This language preserves the structure of the problem in question and the texts in this language can be translated into some representation convenient for computer processing.
- Each reasoning step (in a natural form) from the text under verification must be "obvious" to a computer in the sense that can be checked by it. A checking procedure must evolve for incrementing reasoning steps as much as possible. It must combine general methods of logical inference search with heuristic reasoning techniques such as induction, case reasoning, definition handling, and so on. Such collection of reasoning techniques must also grow and evolve.
- Formal knowledge accumulated in the system (and used in training) must be organized in a hierarchical information environment.

The deductive paradigm is actively investigated in Ukraine from 1990, mainly at the Faculty of Cybernetics of the Kiev State University later renamed as Taras Shevchenko National University of Kyiv.

The SAD System: a Current State

As a result, the System for Automated Deduction (SAD) has been constructed. It can be downloaded or accessed online on the web-site of the Evidence Algorithm project *http://nevidal.org* (see also the papers on SAD: [Lyaletski, 2004], [Anisimov, 2006], [Lyaletski, 2010], [Lyaletski, 2006], [Vershinin, 2000]).

The SAD system conceptually consists of the following components:

- original formal language ForTheL [The Otter, 2012] that is close to the natural English language of mathematical publications; ForTheL texts can be translated into first-order language to allow automated inference search in different logics;
- special module "reasoner" that dispatches a set of traditional proving techniques of mathematical reasoning such as decomposition of a problem, simplification, reasoning by general induction, and others;
- efficient automatic provers: one of them presents the native prover Moses constructed on original sequent-based logical inference search; the other ones are the famous powerful external theorem provers such as, for example, SPASS [The SPASS, 2012], Otter [The Otter, 2012], or Vampire [The Vampire, 2012].

Note that Moses operates in natural language environment exploiting only the signature of an initial theory and, in the case of necessity, has a possibility to use tools for analytical transformations, in particular, some of the tools of the "Analitic-2007" programming system.

The SAD system was used for the formalization and verification of a number of real (non-trivial) mathematical theorems such as Ramsey's Finite and Infinite theorems, Cauchy-Bouniakowsky-Schwarz inequality, Chinese Remainder Theorem and Bezout's Identity (in terms of abstract rings), Tarski's fixed point theorem. Thus, the SAD provides a solid basis for the construction of a deductive testing system.

The following simple example presents a session of testing knowledge of a trainee in Set Theory who knows the ForTheL language. Suppose that after reading the basics of Set Theory, the trainee received the task to prove that a set being a subset of any set is empty. He has a possibility to generate the following ForTheL-text as an input text for the SAD system (containing the proposition to be proved along with its proof and all the necessary definitions, axiom, and several "explanations"):

```
Signature SetSort. A set is a notion. Let S,T denote sets.
Signature ElmSort. An element of S is a notion. Let x belongs to X stand for x is an element of X.
Definition DefSubset. A subset of S is a set T such that every element of T belongs to S.
Definition DefEmpty. S is empty iff S has no elements.
Axiom ExEmpty. There exists an empty set.
Proposition.
    S is a subset of every set iff S is empty.
Proof.
  Case S is empty. Obvious.
  Case S is a subset of every set.
    Take an empty set E.
    Let z be an element of S.
    Then z is an element of E.
    We have a contradiction.
```

Fig. 1. An example of a ForTheL-text to be verified

After checking the above-given text with the help of the SAD system using the Moses prover, the trainee will be able to know that his text is valid (that is, he correctly wrote the proof):

```
[Reason] stdin: verification successful
[Main] sections 22 - goals 6 - subgoals 10 - trivial 1 - proved 5
[Main] symbols 24 - checks 20 - trivial 20 - proved 0 - unfolds 11
[Main] parser 00:00.04 - reason 00:00.00 - prover 00:03.02/00:00.00
[Main] total 00:03.07
```

Fig. 2. The last part of the listing being generated during the SAD session

Analytical transformations in testing

This kind of testing is needed when a solution for an equation must have the form of an analytical (symbolic) expression, for example, a root of an algebraic equation or an equation in partial derivatives. In order to perform such a testing, we need a "shell" that can assure that the symbolic expression proposed by the examinee is correct, that is, it can be transformed into a formula, given by an examiner by means of symbolic computation. Such analytical verification is very appropriate for testing in various domains of physics, trigonometry, algebra, and so on. In the first place, it requires the following generic tools:

- procedures of arbitrary-precision computation for integers, rational and complex numbers;
- methods for determining whether two symbolic expressions are equal; these are usually based on various systems of term rewriting rules;

- methods of term normalization (in particular, normalization to certain conventional mathematical forms);
- tools for analytical transformations of mathematical expressions defined in terms of hierarchical data structures of arbitrary complexity.

The Institute of Problems of Mathematical Machines and Systems of National Academy of Sciences of Ukraine, (IPMMS NASU) started research in this domain in 1960s and created a family of hierarchically developing computer algebra systems in the frame of the "Analitic" project. The specialized computer series "MIR" (Mashina dlya Inzhenernyh Raschetov - Engineering Computation Machine, cf. [Glushkov, 1971],): "MIR-1", "MIR-2", and "MIR-3" having their input languages of the three first version of the "Analitic" language. Later, the developed algorithms were implemented into the SM 1410 computers ("Analitic-79" [Glushkov, 1979]) and into the standard IBM PC ("Analitic-93" [Morozov, 1995] and "Analitic-2000" [Morozov, 2001). Now, the modern project versions called "Analitic-2007" [Morozov, 2007] and "Analitic-2010" [Klimenko, 2010]. are in progress and usage. Let us give a brief description of some of their features and implementation.

Analitic-2007

The "Analitic-2007" version was implemented at the beginning of 2007 [Morozov, 2007]. It inherited all main features of all its predecessors and differed from the previous versions by deeper algebraic transformations, more detailed classification of algebraic tools, sophisticated facilities of calculations control, and improved interactive methods.

The "Analitic-2007" programming system is intended for IBM PCs and is operated as an application for the operating system Windows-98 and higher. It consists of the system kernel and a number of program packages. The compact kernel provides a user with a large quantity of programs, supports the semantic integrity of the "Analitic" language, the universality of its functional properties, and the operability of the "Analitic-2007" system in the environment of the different Windows operating systems. It performs compiling and recompiling programs and data, executing programs, transforming language objects (including programs being considered as objects of the language).

The system automatically determines the size of memory accessible for performing a program and occupies the maximal scope of accessible memory by default. A user has a possibility for determining the size of memory necessary for the normal execution of his programs. In the case of exceeding the existent memory size, the "Analitic-2007" programming system uses virtual memory.

Analitic-2010

The last version "Analitic-2010" [Klimenko, 2010] was significantly changed in its kernel and migrated on the .NET platform. A new user-friendly graphic interface missing in the previous versions was developed.

When implementing "Analitic-2010", the main efforts were directed to the improvement of operating stability of the kernel. For this the parser was recoded without any changes in the language "Analitic". As a result, the software of the previous versions was transferred to the new one.

The new interface is oriented to the efficient handling of data in the interactive mode and the faster generation of new programs. It is equipped with a complete code editor supporting intelligent input and all the possibilities inherent in a modern integrated development environment.

All the "Analitic" family systems are used actively for finding analytical solutions of tasks in mechanics, astronomy, differential equations theory. Besides, a number of experiments were performed in automated analytical transformation in various mathematical learning fields such as, for example, checking algebraic and trigonometric identities.

Combination of deduction and analytical transformations in testing

There exist a great number of software systems for authoring of "electronic textbooks" for various disciplines being taught in secondary schools, colleges, and universities. Their common feature is their orientation towards a broad spectrum of educational branches and, owing to this, towards the simplest kind of examination of trainee's knowledge based on choosing a right answer from a number of alternatives proposed by an examiner. The downside of this technique is that it gives the trainee an incentive to guess a right answer rather than really look for it, which does not allow a tutor to estimate trainee knowledge correctly. A list of "prescribed answers" is notably inconvenient for mathematical disciplines, where a solution to a problem often consists in deriving an analytic (symbolic) expression or in a formal proving when a chain of deductive steps assuring the validity of a statement under consideration must be constructed. Thus, we have the following ways for the computer-aided testing of knowledge obtained by a trainee in the (e-)learning of a subject: the query-answering method, the analytical transformation, the deductive construction, and, their combinations.

The state of the art in automated reasoning and symbolic computation has initiated transition from the simple "choose-an-answer" testing to the more intelligent and complex ones: the deductive and analytical reasoning. As it was mentioned above, the first

approach is useful in studying a mathematical theory allowing its complete formalization for computer checking logical steps of a trainee proving a certain sentence of a theory under consideration (the same concerns any knowledge domain, where formalization and deduction are admissible). The second one is suitable for testing on the base of finding analytical solutions of tasks in algebra, trigonometry, physics, and so on (for both secondary schools and higher education institutions).

We can mention a number of computer proof assistants (for example, Mizar [The Mizar, 2012]. and Isabelle [The Isabelle, 2012], some details can be found in [FmathL, 2012],) as good candidates for using deduction in testing. As to analytical transformation, there exists the great number of computer algebra systems (for example, Mathematica [Wolfram, 2003] and Reduce [20]) one of main purposes of which is to test the correctness of an analytical expression given by a trainee. But in real mathematics, a trainee is faced with texts requiring performing logical steps along with symbolic computation. This leads to the construction of tools for the combination of deduction with analytical transformations.

The last problem can be resolved by means of the "incorporation" of Analitic operators into the FortheL language for using of a linguistic extension in SAD architecture of which was designed in such a way that provided using computer algebra tools in the case of necessity [Verchinine 2007]. Of course, such a reconstruction of ForTheL and SAD will require essential efforts on the consistency of at least data formats of SAD and Analitic, but the authors are sure that moving in this direction will give a new impulse to the improvement of testing a trainee and, as a result, to the appearance on new testing standards and the increasing of e-learning quality.

Extending semantic models for the logical and analytical languages

During the last decade new approach for constructing semantic models for formal languages is being developed at the Faculty of Cybernetics of Taras Shevchenko National University of Kyiv. This approach is called a *composition-nominative approach* [Nikitchenko, 1998]. It aims to construct a hierarchy of program models of various abstraction and generality levels. The main principles of the approach are the following.

- *Development principle* (from abstract to concrete): program notions should be introduced as a process of their development that starts from abstract understanding, capturing essential program properties, and proceeds to more concrete considerations.

- Principle of *priority of semantics* over syntax: program semantic and syntactic aspects should be first studied separately, then in their integrity in which semantic aspects prevail over syntactic ones.
- *Compositionality* principle: programs can be constructed from simpler programs (functions) with the help of special operations, called compositions, which form a kernel of program semantics structures.
- *Nominativity* principle: nominative (naming) relations are basic ones in constructing data and programs.

Here we have formulated only principles relevant to the topic of the article; richer system of principles is developed in [Nikitchenko, 2009].

The above described principles specify program models as *composition-nominative systems* (CNS) [Nikitchenko, 1998]. Such a system may be considered as a triple of simpler systems: 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. Program semantics is considered as partial multi-valued functions over class of data processed by programs; compositions are *n*-ary operations over functions. Thus, composition system can be specified as 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 (descriptive system), and ordinary procedure of term interpretation gives a denotation system.

CNS can be used to construct formal models of various programming, specification, and database languages [Nikitchenko, 1998], [Nikitchenko, 2009]. The program models presented by CNS are mathematically simple, but specify program semantics rather adequately; program models are highly parametric and can in a natural way represent programs of various abstraction levels; there is a possibility to introduce on a base of CNS the notion of special (abstract) computability and various axiomatic formalisms [Nikitchenko, 2001], [Nikitchenko, 2008], [Nikitchenko, 2010].

CNS are classified in accordance with levels of abstraction of their parameters: data, functions, and compositions. For constructing program models three levels of data consideration are chosen: abstract, Boolean, and nominative. At the abstract level data are treated as "black boxes", thus no information can be extracted. At the Boolean level to abstract data new data considered as "white boxes" are added. Usually, these are logical values *T* (true) and *F* (false) from the set *Bool*. At the nominative level data are considered as "grey boxes", constructed of "black" and "white boxes" with the help of naming relations. The last level is the most interesting for programming. Data of this level

are called *nominative data*. The class of nominative data is constructed inductively over a set of names V and a set of basic values W.

Concretizations of nominative data can represent various data structures, such as records, arrays, lists, relations, etc. [Nikitchenko, 1998], [Nikitchenko, 2009]. For example, a set $\{s_1, s_2, ..., s_n\}$ can be represented by a nominative data $[1 \mapsto s_1, 1 \mapsto s_2, ..., 1 \mapsto s_n]$, where 1 is treated as a standard name. Thus, the following data representation principle can be formulated: program data can be represented as concretizations of nominative data.

The above formulated levels of data abstraction may be treated as data intensions. They respectively specify three levels of semantics-based program models: abstract, Boolean, nominative. The models of each level constitute extensions of that level intension. Program models of abstract level are very poor (actually, only sequencing compositions can be defined). Program models of Boolean level are richer and permit to define structured programming constructs (sequence, selection, and repetition). This level is still too abstract and does not explicitly specify data variables. At last, models of nominative level permit to formalise compositions of traditional programming. This level (its intension) involves variables of different types.

Consider, for example, a simple educational programming language WHILE [Nielson, 2009], which is based on three main syntactic components: arithmetic expressions, Boolean expression, and statements. States of WHILE programs can be considered as partial functions from the set V of variables to the set A of basic values and here are denoted by ${}^{V}A$ $(= V \xrightarrow{p} A)$. Thus, semantics of these components is the following: arithmetic expressions specify functions of the type $Fn^{V,A} = {}^{V}A \xrightarrow{p} A$ (called partial quasiary functions), Boolean expressions define functions of the type $Pr^{V,A} = {}^{V}A \xrightarrow{p} Bool$ (partial quasiary predicates), statements specify functions of the type $Prg^{V,A} = {}^{V}A \xrightarrow{p} {}^{V}A$ (partial biquasiary functions). Note that ${}^{V}A$ is a class of single-valued nominative data. Functions over nominative data are called nominative functions. Main operations over nominative data with the name v as a parameter are naming, denaming, and checking. The main compositions (assignment, sequential, conditional, while compositions) can be formally defined over nominative functions. Obtained CNS formalize semantics of simple programming languages. Formalization of more complex languages requires more powerful classes of nominative data (hierarchic, with complex names, indirect naming, etc.) and more powerful compositions (recursive, concurrent, etc).

CNS can specify the main aspects of programming languages, and, as a consequence, it is possible to construct e-learning tools that support studying of various aspects of programming.

Based on described composition-nominative program models of various abstraction levels new logics which correspond to such models were developed. Such logics were called *composition-nominative logics* (CNL) and are oriented on program reasoning. They are logics of partial quasiary predicates and functions. Their compositions are derived from Kleene's strong connectives that permit to work with partial predicates.

Three kinds of logics can be constructed based on composition-nominative program models:

- logics, which use only partial quasiary predicates (pure predicate logic);
- logics, which use additionally partial quasiary functions (predicate-function logics);
- logics, which use also biquasiary functions (program logics).

The first type of logics generalizes classical pure predicate logics, the second type corresponds to classical predicate logic (with functions and equality), and the third type can present various logics, which use program constructs.

The following classification of these kinds of logics was proposed.

For logics of pure quasiary predicates we identify renominative, quantifier, and quantifier-equational levels.

Renominative logics are the most abstract among the above-mentioned logics. The main composition for these logics is the composition of renomination (renaming), which is a total mapping $R^{v_1,\ldots,v_n}_{x_1,\ldots,x_n} : Pr^{V,A} \xrightarrow{t} Pr^{V,A}$. Intuitively, given a quasiary predicate P and a nominative set d, the value of $R^{v_1,\ldots,v_n}_{x_1,\ldots,x_n}(P)(d)$ is evaluated in the following way: first, a new nominative set d' is constructed from d by changing the values of the names $v_1,\ldots,v_n$ in d to the values of the names $x_1,\ldots, x_n$ respectively; then predicate P is applied to d'. The obtained value of P (if it was evaluated) will be the result of $R^{v_1,\ldots,v_n}_{x_1,\ldots,x_n}(P)(d)$. For simplicity's sake the notation $R^{\bar{v}}_{\bar{x}}$ for renomination composition is also used. The basic composition operations of renominative logics are $\vee$, $\neg$, and $R^{\bar{v}}_{\bar{x}}$.

At the *quantifier* level, all basic (object) values can be used to construct different nominative sets to which quasiary predicates can be applied. This allows one to introduce the compositions of quantification $\exists x$ in style of Kleene's strong quantifiers. The basic compositions of logics of the quantifier level are $\vee$, $\neg$, $R^{\bar{v}}_{\bar{x}}$, and $\exists x$.

At the *quantifier-equational* level, new possibilities arise for equating and differentiating values using special 0-ary compositions, i.e., parametric equality predicates $=_{xy}$. Basic compositions of logics of the quantifier-equational level are $\vee$, $\neg$, $R^{\bar{v}}_{\bar{x}}$, $\exists x$, and $=_{xy}$.

All specified logics (renominative, quantifier, and quantifier-equational) are based on algebras which have only one sort: a class of quasiary predicates.

For quasiary predicate-function logics we identify function and function-equational levels.

At the *function* level, extended capabilities of formation of new functions and predicates are obtained. At this level it is possible to introduce the superposition composition $S^{\bar{x}}$ (see [Nikitchenko, 2008]), which formalizes substitution of functions into predicate (or function). It also seems natural to introduce special 0-ary compositions, called denaming functions 'x. Given a nominative set, 'x yields a value of the name x in this set. Introduction of such functions allows one to model renomination compositions with the help of superposition. The basic compositions of logics of the function level are $\vee$, $\neg$, $S^{\bar{x}}$, $\exists$x, and 'x.

At the function-equational level a special equality composition = can be introduced additionally. The basic compositions of logics of the function-equational level are $\vee$, $\neg$, $S^{\bar{x}}$, $\exists x$, *'x*, and = . At this level different classes of first-order logics can be defined.

This means that two-sorted algebras (with sets of predicates and functions as sorts and above-mentioned compositions as operations) form a semantic base for first-order CNL.

The level of *program logics* is quite rich. First, program compositions should be defined that describe the structure of programs. In the simplest case of structured programming these are:

- assignment composition AS^x: $Fn^{V,A} \xrightarrow{t} Prg^{V,A}$,
- composition of sequential execution $\bullet$: $Prg^{V,A} \times Prg^{V,A} \xrightarrow{t} Prg^{V,A}$,
- conditional composition *IF*: $Pr^{V,A} \times Prg^{V,A} \times Prg^{V,A} \xrightarrow{t} Prg^{V,A}$,
- cycling composition *WH*: $Pr^{V,A} \times Prg^{V,A} \xrightarrow{t} Prg^{V,A}$.

Let us note that above presented logics of partial predicates can be considered generalizations of classical logics. First of all, this concerns types of predicates: while classical logic is semantically based on total *n*-ary predicates, CNL are based on partial quasiary predicates, defined on a special type of nominative data. For such logics valid and complete sequent calculi were constructed [Nikitchenko, 2008]. More complex CNL are defined over hierarchic nominative data. Importance of such data is explained by their representational power that permits to model data structures of specification and programming languages. Characteristic feature of such languages is usage of composite names to access data components. The constructed logics also use composite names. On the next generalization steps modal and temporal CNL are defined and investigated [Nikitchenko, 2008].

Concerning the educational aspects of the proposed approach to formal languages specification, we can admit that this approach permits to integrate on one methodological and mathematical basis such disciplines as programming, mathematical logic, and

computability theory [Nikitchenko, 2010]. The integration is based on the idea that all these disciplines have as their kernel the notion of a specialized language system. Integration of such disciplines can be achieved by

- usage of common methodological construction principles of such disciplines;
- uniform development of main notions of disciplines;
- construction of uniform formal models of the main notions;
- constructing of the uniform e-learning tools.

This approach seems to be useful in e-learning systems because

- it is based on a small number of universal methodological principles applicable to different discipline;
- it widely uses the principle of development which proposes a number of levels starting from simple to more elaborate thus giving possibility to present more complex concepts on later stages of teaching;
- it leads to simple formal language models thus permitting their thorough investigation with further implementation of e-learning tools.

So, the constructed formal models of programming and logical languages permit to extend possibilities for deduction tools developed in Kiev by including a program reasoning component. Such extension will usually require transformation of CNL formulas into first-order classical logic [Nikitchenko, 2012].

Conclusion

The above-given analysis of Kiev approaches to symbolic computation and deduction demonstrates that the advances made by researches at IPMMS and Taras Shevchenko National University of Kyiv allow introducing and implementing various forms of distant e-learning based on a more thorough and unbiased evaluation of an examinee, which can improve the quality of learning for disciplines which admit (at least partial) formalization. Moreover, integration of analytical and deductive testing in a common framework (say, within intelligent tutoring systems) based on extended logical languages allow these two forms of intelligent testing to complement and enforce each other. Constructed tools can be incorporated into the existing e-learning systems taking into account the specifics of a domain under study. Also, one can use the proposed framework to design and implement electronic courses and textbooks, containing learning material as well as exercises for simple and intellectual testing for objective evaluation of student's knowledge.

Bibliography

[Glushkov, 1970] Glushkov V.M. Some problems of automata theory and artificial intelligence (in Russian). Kibernetika, No. 2, 1970, P. 3–13.

[Lyaletski, 2004]. Lyaletski, A., Paskevich, A., Verchinine, K.: Theorem proving and proof verification in the system SAD. In Asperti, A., Bancerek, G., Trybulec, A., eds.: Mathematical Knowledge Management: Third International Conference, MKM-04, Volume 3119 of Lecture Notes in Computer Science, Springer, 2004, P. 236–250.

[Anisimov, 2006]. Anisimov A. V. and Lyaletski A. V., (2006). The SAD system in three dimensions, Proceedings of the SYNASC'06, Timisoara, Romania, 2006, P. 85-88.

[Lyaletski, 2010]. Lyaletski A. and Verchinine K. Evidence Algorithm and System for Automated Deduction: A retrospective view. Intelligent Computer Mathematics: 10th International Conference AISC/Calculemus/MKM 2010 (Paris, France, July 2010), Vol. 6167 of Lecture Notes in Computer Science, Springer-Verlag, 2010, P. 411-426.

[Lyaletski, 2006]. Lyaletski, A., Paskevich, A., Verchinin, K.: SAD as a mathematical assistant — how should we go from here to there? Journal of Applied Logic, Vol. 4(4), 2006, P. 560–591.

[Vershinin, 2000]. Vershinin, K., Paskevich, A.: ForTheL — the language of formal theories. International Journal of Information Theories and Applications, Vol. 7(3), 2000, P. 120–126.

[The SPASS, 2012] The SPASS Prover: http://www.spass-prover.org/.

[The Otter, 2012]. The Otter automated deduction system: http://www.mcs.anl.gov/research/projects/AR/otter/.

[The Vampire, 2012], The Vampire prover: http://www.vprover.org/.

[Glushkov, 1971], Glushkov V. M., Bodnarchuk V. G., Grinchenko T. A., Dorodnizyna A. A., Klimenko V. P., Letichevsky A. A., Pogrebinsky S. B., Stogniy A. A., and Fishman Yu. S. ANALITIK (an algorithmic language for description of computational processes using analytical transformations) (in Russian), Kibernetika, No.3, 1971, P. 102-134.

[Glushkov, 1979], Glushkov V. M., Grinchenko T. A., Dorodnizyna A. A., Drakh A. M., Klimenko V. P., Pogrebinsky S. B., Savchak O. N., Fishman Yu. S., and. Tsaryuk N. P. ANALITIK-79 (in Russian), Technical report, Institute of Cybernetics, Kiev, USSR, 1979.

[Morozov, 1995]. Morozov A. A., Klimenko V. P., Fishman Yu. S., Bublik B. A., Gorovoy V. D., and Kalina E. A. ANALITIK-93 (in Russian), Kibernetika i sistemnyj analiz, No.5, 1995, P. 127-157.

[Morozov, 2001]. Morozov A. A., Klimenko V. P., Fishman Yu. S., Lyakhov A. L., Kondrashov S.V., and Shvalyuk T. N. ANALITIK-2000 (in Russian), Matematicheskie mashiny i sistemy, No. 1-2, 2001, P. 66-99.

[Morozov, 2007]. Morozov A. A., Klimenko V. P., Fishman Yu. S., and Shvalyuk T. N. ANALITIK-2007 (in Russian), Mathematical Machines and Systems, No. 3-4, 2007, P. 8-52.

[Klimenko, 2010]. Klimenko V. P., Lyakhov A.L., Gvozdik D.N., Zakharov S.A., and Shvalyuk T. N. On the implementation of a new version of the Analitic family CAS (in Russian), Proceedings of the International conference CMSEE-2010, Poltava, 2010.

[The Mizar, 2012], The Mizar system: http://www.mizar.org/.

[The Isabelle, 2012], The Isabelle system: http://www.cl.cam.ac.uk/research/hvg/Isabelle/.

[FmathL, 2012], FMathL - Formal Mathematical Language: http://solon.cma.univie.ac.at/FMathL.html

[Wolfram, 2003], Wolfram S., (2003). The Mathematica Book, Fifth Edition, Wolfram Media, Inc.

[Reduce, 2012], The computer algebra system Reduce: http://reduce-algebra.sourceforge.net/.

[Verchinine 2007], Verchinine, K., Lyaletski, A., and Paskevich, A. System for Automated Deduction (SAD): a tool for proof verification. In Automated Deduction, 21st International

Conference, CADE-21 (Bremen, Germany, July 2007), F. Pfenning, Ed., vol. 4603 of Lecture Notes in Computer Science, Springer-Verlag, P. 398-403.

[Nikitchenko, 1998], Nikitchenko, N.S.: A Composition Nominative Approach to Program Semantics. Technical Report IT–TR 1998-020, Technical University of Denmark, 103 p., 1998.

[Nikitchenko, 2009], Nikitchenko M.S., Composition-nominative aspects of address programming, Kibernetika I Sistemnyi Analiz, 2009, 6, P. 24-35 (In Russian)

[Nielson, 2009], Nielson H.R., Nielson F.: Semantics with Applications: A Formal Introduction. John Wiley & Sons Inc, 1992.

[Nikitchenko, 2001]. Nikitchenko N.S., Abstract computability of non-deterministic programs over various data structures, In: Bjørner D., Broy M., Zamulin A.V. (Eds.), Perspectives of System Informatics, LNCS, 2001, 2244, P. 471-484.

[Nikitchenko, 2008]. Nikitchenko M.S., Shkilnyak S.S., Mathematical logic and theory of algorithms, Publishing house of Taras Shevchenko National University of Kyiv, Kyiv, 2008, 528 p. (in Ukrainian)

[Nikitchenko, 2010]. Nikitchenko M.S. Integrating programming-related disciplines: main principles and notions. In: Proc. of 8th Int. Conference on Emerging eLearning Technologies and Applications. The High Tatras, Slovakia, October 28-29, 2010, P. 49–56.

[Nikitchenko, 2012]. Nikitchenko M.S., Tymofieiev V.G.: Satisfiability Problem in Composition-Nominative Logics of Quantifier-Equational Level. In: Proc. 8-th Int. Conf. ICTERI 2012, Kherson, Ukraine, June 6-10, 2012. CEUR-WS.org/Vol-848, P. 56-70.

Authors' Information

Vitaly Klimenko *– Deputy Director of the Institute of Problems of Mathematical Machines and Systems of NAS of Ukraine, 42, Acad. Glushkova Ave., 03680, Kyiv, Ukraine;*

e-mail: klimenko@immsp.kiev.ua

Major Fields of Scientific Research: Computer algebra, Mathematical software, Architecture of computer systems

Alexander Lyaletski *– Senior researcher of the Faculty of Cybernetics at the Taras Shevchenko National University of Kyiv, 64, Volodymyrska Street, 01601 Kyiv, Ukraine;*

e-mail: lav@unicyb.kiev.ua

Major Fields of Scientific Research: Automated Reasoning, Proof theory, Mathematical and Applied Logics

Mykola Nikitchenko *– Chairman of the department of theory and technology of programming at the Taras Shevchenko National University of Kyiv, 64, Volodymyrska Street, 01601 Kyiv, Ukraine; e-mail: nikitchenko@unicyb.kiev.ua*

Major Fields of Scientific Research: Foundations of informatics, Formal software system development, Mathematical logic, Computability theory, Courseware for informatics