Saint-Petersburg State University

Alexander Sergeevich Gerasimov

Design and implementation of a proof search algorithm for an extension of infinite-valued predicate Lukasiewicz logic

Abstract
of the PhD thesis in Computer Science

Saint-Petersburg – 2007

This research was carried out at Computer Science Department, Faculty of Mathematics and Mechanics, Saint-Petersburg State University.

Scientific adviser: Nikolai Kirillovich Kossovski, Doctor of Science, professor

Official opponents: Vladimir Pavlovich Orevkov, Doctor of Science
Artem Valerievich Tishkov, PhD

Organization acting as an opponent:
Saint-Petersburg State Polytechnical University

The defence of the thesis is on 26th of April, 2007 at 15:30 on a meeting of Dissertation Council 212.232.51 of Saint-Petersburg State University.

General description of the thesis

Urgency of the research. A growing interest in many-valued (in particular, infinite-valued) logics is based on their various applications that include representation of fuzzy knowledge and approximate reasoning (see, e.g., [12, 14]).

As a field of mathematical logic, many-valued logic has been developed in parallel with fuzzy logic, which goes back to L. Zadeh [18, 3]. Fuzzy logic of Zadeh is based on theory of fuzzy sets. A fuzzy set is a set with fuzzy bounds, more formally such a set is defined by means of a membership function that assings a real number from the segment $[0, 1]$ to an element. *Fuzzy logic in broad sense* (or simply *fuzzy logic*) is a discipline that uses notions of fuzzy set theory to develop methods of applied approximate reasoning (see [15]). Fuzzy logic is employed in industrial systems of fuzzy control, such as household appliances. However, methods of fuzzy logic are not well-founded from the point of view of formal logic.

Formalization of fuzzy logic is activly pursued in the last decade (see the fundamental books [14, 12, 7]). In connection with this formalization, *mathematical fuzzy logic* (or *fuzzy logic in narrow sense*) is identified as a discipline that developes deductive systems for fuzzy logic so that fuzzy logic is treated as rigorous mathematical many-valued logic. Infinite-valued predicate Lukasiewicz logic (its description can be found, e.g., in [14]) is used to formalize fuzzy logic.

The process of formation of mathematical fuzzy logic is far from its completion, because fuzzy logic uses concepts that have no analogs in many-valued logic. Such a concept is linguistic modifiers like "very", "extremely", "quite", etc. L. Zadeh [3] squares a membership function to represent modifier "very", this makes formalization of fuzzy logic hard. Therefore an alternative and simpler formalization of linguistic modifiers is an important step towards approaching mathematical fuzzy logic and fuzzy logic in broad sense; moreover, such a formalization would extend applications of mathematical fuzzy logic.

Proof search methods suitable for automation with computers are needed for many successful applications of a logic. Before this research, only Hilbert-type calculi were known for infinite-valued predicate Lukasiewicz logic (these calculi can be found, e.g., in [14, 12]). It is widely accepted that Hilbert-type calculi are not suitable for automatic proof search.

For infinite-valued propositional Lukasiewicz logic, there are various proof search methods, among them: sematic tableaux [13, 17], sequent calculi [5, 10, 16]. We especially note the sequent calculus for a so-called level logic [5]. Logical connectives of the level logic allows to represent formulas of Lukasiewicz logic. Axiom recognition is performed by means of linear programming methods. This approach is developed in the present research.

Attempts of applying proof search methods for infinite-valued propositional Lukasiewicz logic to approximate reasoning are awkward, because, in particular,

these methods do not cover predicate logic. For example, predicate formulas have to be translated to propositional ones to representat fuzzy knowledge in the recent paper [16]. Firstly, such a translation can be performed only for finite domains of predicates and, secondly, it essentially lengthens formulas that represent source knowledge.

Thus development of calculi, which are suitable for proof search, for infinite-valued predicate Lukasiewicz logic and implementation of computer programs for proof search are challenging research tasks. Moreover, the increasing interest in infinite-valued predicate Lukasiewicz logic in connection with development of mathematical fuzzy logic stipulate automation of proof search for this logic of Lukasiewicz.

Goals of the research. To extend infinite-valued predicate Lukasiewicz logic with means for expressing modifiers like "very", to develop a calculus for the extension, and to implement a proof search algorithm in the calculus.

Main results

1. A sequent calculus for infinite-valued predicate Lukasiewicz logic extended with linguistic modifiers like "very" was formulated.

2. Properties of the proposed sequent calculus were investigated, these properties provide a theoretical basis for development of a proof search algorithm in the calculus.

3. An algorithm for proof search in the proposed calculus was developed. Correctness of the algorithm was proved.

4. The proof search algorithm was implemented as an application programming interface.

5. An algorithm for solving systems of linear two-term inequalities was refined, this algorithm is used to recognize some axioms of the proposed calculus. Time complexity of the algorithm was evaluated in a formal computational model.

6. The algorithm for solving systems of linear two-term inequalities was implemented as an application programming interface.

Scientific novelty. All the main results of the thesis are novel. Before this research, only Hilbert-type calculi for infinite-valued predicate Lukasiewicz logic were known and neither theoretical basis nor software for automatic proof search in this logic were developed.

Theoretical and practical value. The proposed logic can be used to represent fuzzy knowledge. The formulated sequent calculus can be used for proof search in infinite-valued predicate Lukasiewicz logic as well as its proposed extension. Proof search in the sequent calculus is significantly more effective than proof search in a Hilbert-type calculi.

The implemented application programming interface (API) for proof search can be employed, for example, in research for automatic proof search in infinite-valued predicate Lukasiewicz logic and its proposed extension. The API can serve as an inference engine of a deductive system based on either of the logics in question.

The implemented API for solving systems of linear two-term inequalities can be used to solve problems, which size is significantly greater than size of problems that can be solved with popular computer algebra systems.

Approbation of the thesis. The results of the thesis were presented on

- VIII and IX All-Russian scientific conferences "Contemporary logic: problems of theory, history and applications in science" (Saint-Petersburg, 2004 and 2006);
- Contest-conference for students, PhD students and young scientists of North-West "Microsoft technologies in theory and practice of programming" (Saint-Petersburg, 2005);
- International conference "Stability and control processes" (Saint-Petersburg, 2005);
- a seminar of Saint-Petersburg department of Russian association of artificial intelligence (Saint-Petersburg, 2006);
- XVI International school-seminar "Synthesis and complexity of control systems" (Saint-Petersburg, 2006);
- Tenth national conference on artificial intelligence CAI-06 (Obninsk, 2006).

Publications. The main results were published in papers [$1'$ – $6'$].

Structure and volume of the thesis. The thesis consists of 6 chapters, references and 2 appendices. The volume of the thesis is 194 pages. The principal contents of the thesis take 168 pages, appendices take 26 pages. The references consist of 82 items.

Contents of the thesis

The **first chapter** contains a short abstract of the thesis, a description of infinite-valued predicate Lukasiewicz logic, an overview of related papers, and a justification of urgency of the research. Then goals of the research are posed and brief contents of the next chapters are given.

The **second chapter** is devoted to a description of a logic proposed (the logic is denoted by Lq), a sequent calculus LqS for the logic, and properties of the calculus [2′, 1′].

A language of the logic Lq and its semantics are defined in the first section of this chapter. An integral part of a predicate variable is a so-called its *segment of truth values* $[a, b]$, where a, b are rational numbers, $a < b$. A term is an individual variable or an individual constant. An *atomic formula* is a rational number, a propositional variable, or a predicate variable followed by a bracketed list of terms. A *formula* of the logic Lq is an atomic formula or $(A\&B)$, $(A \vee B)$, $(A \prec B)$, $q \cdot A$, $\forall x A$, or $\exists x A$, where A and B are formulas of the logic Lq, q is a rational number, x is an individual variable. Connectives $\prec$ and $q\cdot$ are called *fuzzy inequality* and a *moderator* respectively.

In order to define semantics of a language of logic Lq, notions of *inerpretation* and *evaluation* of the language are introduced. These notions are similar to classical ones only that here an interpretation takes each predicate variable to the predicate, which range of values is a subset of the segment of truth values of the predicate variable (an interpretation takes a propositional variable to the real number that belongs to the segment of truth values of the propositional variable). If an interpretation and an evaluation of a language are specified, then a formula A is assigned its *truth value* $[A]$, which is a real number, according to the following rules: $[(A\&B)] = \min([A], [B])$, $[(A \vee B)] = \max([A], [B])$, $[(A \prec B)] = [B] - [A]$, $[q \cdot A] = q \cdot [A]$, $[\forall x A] = \inf_x [A]$, $[\exists x A] = \sup_x [A]$. A formula is called *valid*, if its truth value is nonnegative in any interpretation and any evaluation.

Note that, firstly, moderators can increase or decrease truth values of formulas, therefore linguistic modifiers like "very" can be formalized by means of moderators. Secondly, suppose A is an Lq formula, r is a rational number; then the statement 'A takes on truth values greater or equal to (respectively, less or equal to) r in any interpretation and any evaluation' is equivalent to '$(r \prec A)$ (respectively, $(A \prec r)$) is valid'. Thirdly, any formula of infinite-valued predicate Lukasiewicz logic can be represented as a formula of the logic Lq so that truth values of these formulas are equal in any interpretation and any evaluation.

The following example of formalization of approximate reasoning using the logic Lq is given. Premises 'if an object is small, then it is difficult to discern the object' and 'the object z is very small' imply that 'it is quite

difficult to discern the object z'. Let us introduce a predicate $P1[0,1](x)$ that take an object x to the real number from the segment $[0,1]$, the number expresses the degree of smallness of the object, and a predicate $P2[0,1](y)$ that similarly expresses the degree of difficulty of discerning an object y. We formalize the modifier "very" by means of the moderator $1/2\cdot$ and the modifier "quite" by means of the moderator $2/3\cdot$. We represent the given approximate reasoning as a formula of the logic Lq: $((\forall x(P1[0,1](x) \prec P2[0,1](x)) \,\&\, 1/2 \cdot P1[0,1](z)) \prec 2/3 \cdot P2[0,1](z))$. Thus in order to justify this reasoning, it is sufficient to prove the appropriate formula.

In the end of the first section of the second chapter, a theorem about nonenumerability of the set of all valid formulas of the logic Lq is proved.

In the second section of the second chapter, an antecedent-free sequent calculus LqS for the logic Lq is formulated and concomitant notions are defined. A *sequent* is a finite list of Lq formulas separated with commas (each of these formulas is called a *member* of the sequent), some formulas may be repeated, an order of formulas in the list is of no importance. A sequent is called *valid*, if the disjunction of all its members is valid (the empty sequent is represented by the number -1). *Inference rules* are listed, the rules introduce logical symbols exept for fuzzy inequality and a moderator that stands just in front of an atomic formula (formulas that contain only atomic formulas with moderators in front of them and fuzzy inequalities are processed on axiom recognition).

Then axioms of the calculus LqS are defined. A *canonical chain of inequalities (CCI)* is defined in the following way. Each formula of the form P or $q \cdot P$ (where $q\cdot$ is a moderator, P is an atomic formula) is a CCI. If I is a CCI and J is a CCI, then $(I \prec J)$ is a CCI.

Let S be a sequent. Suppose each member of S that is not a CCI is eliminated from S; then the sequent obtained is called the *basic subsequent* of the sequent S. We say that a sequent is an *axiom* if its basic subsequent is valid.

The algorithm given in the thesis to each sequent with the nonempty basic subsequent assigns the system of strict and nonstrict linear inequalities with rational coefficients and rational-valued variables. The following theorem is proved: such a sequent is an axiom iff the corresponding system of linear inequalities is inconsistent.

Some properties of the calculus LqS are proved in the third section of the second chapter. Let us list the most important properties.

1. All the inference rules and their inversions keep validity of sequents. The calculus LqS is sound.

2. The calculus LqS is consistent.

3. The antecedent-free sequent calculus for classical two-valued logic can be embedded into the calculus LqS.

4. A sound and complete calculus for the logic Lq does not exist.

5. The calculus LqS is undecidable.

6. The calculus LqS is complete for the propositional fragment of the logic Lq.

7. The propositional fragment of the calculus LqS is decidable.

8. Minus-normalization is admissible in a proof search in the calculus LqS.

Let us explain the last property. When we search for a proof of a sequent S *bottom-up*, we find sequents that are premises of an application of an inference rule such that S is the conclusion of the application, then for each sequent S' obtained, if S' is not an axiom, we find sequents that are premises of an application of an inference rule such that S' is the conclusion of the application, and so on. Thus we *apply* inference rules *backward*. On a backward application of an inference rule that introduces a given occurrence of a logical symbol, premises are constructed in a determinate way except for the case when a so-called *existential* rule is applied backward. Each existential rule of the calculus LqS resembles the rule of existential quantifier introduction into the succedent of a sequent in Gentzen sequent calculus for classical two-valued logic. There are infinitely many variants of choosing a witnessing term on a backward application of an existential rule of the calculus LqS.

For classical two-valued first-order logic, sequent calculi are known (see, e.g., [4] as well as [6, 11]) such that the searching of witnessing terms on a bacward application of an existential rule is finite. Such an elimination of infinite searching of terms is called *minus-normalization* [6]. However, the mentioned papers (and other publications we are aware of) do not contain a proof of equivalence of their sequent calculi and a conventional sequent calculus for classical two-valued logic.

In the present thesis, we formulate a restriction on witnessing terms (a witnessing term is one of the terms that occur in the conclusion of an existential rule) and prove that this restriction does not change the set of provable sequents. (Then minus-normalization for the sequent calculus for classical two-valued logic is justified by property 3.)

In the fourth section of the second chapter, a sublogic $Lq2$ of the logic Lq and a sequent calculus $Lq2S$ for the sublogic are described. Use of the connective $\prec$ in formulas of the sublogic $Lq2$ is restricted so that recognizing of an axiom of the calculus $Lq2S$ is reduced to testing inconsistency of the system of linear inequalities, each such an inequality has no more than two terms. There exists a strongly polynomial algorithm to test inconsistency of such systems (such an algorithm is described in the fifth chapter of the thesis), whereas only polynomial algorithms for axiom recognition in the calculus LqS are known. The sublogic $Lq2$ appear sufficiently expressive for initial modeling of a fuzzy knowledge domain.

In the **third chapter**, a proof search algorithm is described and its properties are proved.

When a proof of a sequent S is searched bottom-up, a *proof search tree* is constructed naturally: the root of the tree is the sequent S, the immediate descendants of the root are sequents, which are premises of a backward application of an inference rule to the sequent S, and so on (the tree is considered as growing upwards from the root). A proof search tree becomes a *proof tree* (i.e., the proof is found) when all leaves of the tree are axioms.

Despite the fact that infinite searching of witnessing terms can be avoided, the problem of choosing witnessing terms needs to be addressed still. We use the *method of metavariables*: we defer the choice of a concrete witnessing term on a backward application of an existential rule and substitute a unique metavariable in place of such a term. Thus a *proof skeleton* is constructed instead of a proof search tree. Then sometimes we are to check whether we can assign terms to metavariables so that the proof skeleton becomes the proof tree.

The proposed algorithm *Prove* [3′] searches for a proof of a given sequent constructing a proof skeleton. Whenever an existential rule is applied backward and a metavariable is introduced, the *substitution set* is associated with the metavariable, this finite set contains all terms that are sufficient to substitute for the metavariable according to the restriction of minus-normalization. Then in order to turn a proof skeleton to a proof tree, *unification* is performed, i.e., substitution sets of all metavariables of the skeleton are searched for values of metavariables so that the skeleton becomes the proof tree.

An idea of the algorithm is discussed and notions in use are defined in the first section of the third chapter.

In the second section, steps of the main algorithm *Prove* and auxiliary algorithms (an algorithm for axiom recognition and a unification algorithm) are described. Also requirements to an auxiliary algorithm called *proof search tactics* are stated. Given a proof skeleton, such an algorithm reports, when unification is to be performed, and chooses (a) a leaf sequent S, (b) an inference rule (R), which backward application are to be performed, and (c) an occurrence of the logical symbol in S such that the occurrence can be introduced into S with an application of the rule (R). An example of a proof search with comments is listed in the end of the second section.

In the third section, some properties of the auxiliary algorithms and the main algorithm *Prove* are proved. Finally the following theorem is proved.

Theorem. *Suppose the algorithm Prove uses any proof search tactics; S is a sequent, which is the input data for the algorithm Prove. Then the following statements are true.*

(1) If the algorithm Prove gave the answer "provable", then the sequent S is provable in the calculus LqS.

(2) If the algorithm Prove gave the answer "unprovable", then the sequent S is unprovable in the calculus LqS.

(3) Suppose the sequent S does not contain a quantifier. Then the algorithm Prove gives the answer "provable" if the sequent S is provable in the calculus LqS, and gives the answer "unprovable" if the sequent S is unprovable in the calculus LqS.

In the end of the third section, a choice of proof search tactics is discussed and one tactics is described. The tactics chooses existential rules for backward applications "uniformly", giving different occurrences of quantifiers equal opportunities to participate in backward applications.

In the **fourth chapter**, a software implementation of the proof search algorithm is described [3′]. The algorithm is implemented in the Java programming language as an application programming interface (API), which provides programmatic interface to access its functionality. In the beginning of the chapter, the purpose of main classes is breifly described and class diagrams, which represent hierarchies of logical symbols, terms, and formulas, are given.

Then a policy of object sharing in the program representation of formulas (using syntax trees with possibly shared leaves) and sequents is specified. The policy, on the one hand, allows to uniquely and efficiently identify an occurrence of a non-atomic subformula in a formula and, on the other hand, safes a lot of memory as formulas may be shared between sequents.

Then main features of the software implementation of the proof search algorithm are described. Let us mention the following key features.

Any proof search tactics is defined through an interface `Tactics`. The main algorithm for proof search can employ any tactics that implement this interface. (Such an architecture is known as the design pattern *strategy* [8].)

Each inference rule is represented as an object, which has a method that applies the rule backward. The main algorithm of proof search delegates a backward application of a rule, which is chosen by the tactics, to the rule itself. Thus the inference rules can be easily modified.

Every system of linear inequalities, which is constructed on axiom recognition, is tested for inconsistency by an auxiliary algorithm described in the fifth chapter of the thesis if all the inequalities are two-term, otherwise the system of linear inequalities is tested by the function `FindInstance` of the computer algebra system Mathematica (via J/Link, a toolkit that links a Java program and Mathematica).

A systematic survey of the public API for proof search is given in the end of the fourth chapter.

The volume of the program source code written, including the implementation of the algorithm for solving systems of linear two-term inequalities, is about 9000 lines.

The **fifth chapter** is devoted to an algorithm for testing consistency of systems of strict and nonstrict linear two-term inequalities with integer coef-

ficients and rational-valued variables as well as an algorithm for solving such systems (an algorithm of solving systems is an algorithm that finds at least one solution of the system if a solution exists, otherwise the algorithm reports that the system is inconsistent). These algorithms are based on the method of elimination of variables and removal of redundant inequalities so that the number of inequalities that contain any two variables is limited by a constant fixed beforehand.

An algorithm is said to be *strongly polynomial* (see, e.g., [9]), if (a) it is polynomial in time on Turing machine, and (b) a number of elementary arithmetic operations (addition, subtraction, multiplication, division, comparison), which are performed by the algorithm on rational numbers, is bounbed by a polynomial of the number of integers in an input.

These algorithms were proposed in [2] and the condition (b) of the above definition was established ibidem.

In the present thesis, these algorithms are refined, moreover, some steps are added to the algorithms (the description of the algorithms from [2] is incorrect without these steps) and an auxiliary algorithm for removal of redundant inequalities is developed (see [5′, 6′]). Correctness of the algorithms described in the thesis is proved, i.e., the algorithm for testing consistency of systems of linear two-term inequalities gives the answer "consistent" if an input system is consistent, and gives the answer "inconsistent" if an input system is inconsistent. (an analogous statement is proved for the algorithm for solving systems).

Then a computational model, which is close to random access machine with logarithmic cost criterion (see [1]) is specified and a polynomial bound on time complexity of the described algorithms is obtained [6′]. It is established that the algorithms are polynomial in time on Turing machine, this allows to prove the algorithms are strongly polynomial.

In the last section of the fifth chapter, an implementation of the algorithms in the Java programming language as an application programming interface is described [4′]. There are two algorithms with similar steps in the implementation: the mentioned algorithm for solving systems with removal of redundant inequalities and an algorithm based on the ordinary method of elimination of variables. One of the principal tasks of object-oriented programming, common behavior extraction for the purpose of code reuse, is accomplished by means of *template method* [8].

Finally, results of experiments comparing performance of the algorithm for solving systems, which is implemented by the author of the thesis, and an algorithm, which is represented by the function `FindInstance` of the computer algebra system Mathematica, are given. The experiments were carried out on a personal computer. To solve a system, which amounts several thousands of inequalities, the implemented algorithm spends several seconds, whereas Mathematica spends several tens of minutes. To solve a system, which amounts

several tens of thousands of inequalities, the implemented algorithm spends up to several tens of seconds, whereas Mathematica does not finish its work in 12 hours. These results show that the implemented algorithm is significantly more efficient for solving systems of linear two-term inequalities than the algorithm used in the computer algebra system Mathematica.

The **sixth chapter** contains a list of main results of the thesis.

In the **appendix A**, a source code of a short program, which uses the implemented API for proof search, is given. A report of a proof search of a formula, listed in the next section of this appendix, was produced by the program. An example of formalizing fuzzy knowledge represented as natural language sentences using the logic Lq and deducing new fuzzy knowledge from the initial facts is described in the last section of the appendix A.

The **appendix B** contains a source code of a short program, which makes use of the implemented API for solving systems of linear two-term inequalities and is used for comparing performance of the algorithm for solving systems implemented by the author of the thesis and the algorithm, which is represented by the function `FindInstance` of the computer algebra system Mathematica.

References[1]

[1] A. V. Aho, J. E. Hopcroft, J. D. Ullman. The Design and Analisys of Computer Algorithms. Addison-Wesley, 1976.

[2] D. V. Davydok. On consistency of systems of two-term linear inequalities. In: N. K. Kossovski, A. V. Tishkov. Logics of finite-valued predicates based on inequalities. Saint-Petersburg University, 2000. Pp. 246–268. (In Russian.)

[3] L. A. Zadeh. The Concept of a Linguistic Variable and its Application to Approximate Reasoning. American Elsevier Publishing Company, 1973.

[4] S. Kanger. A simplified proof method for elementary logic. In: Computer programming and formal systems. North-Holland, Amsterdam, 1963. Pp. 87–93.

[5] N. K. Kossovski. Level logics. In: Proceedings of scientific seminars of Petersburg Dept. Math. Inst. RAS. Vol. 220, 1995. Pp. 72–82. (In Russian.)

[1]The first 8 items are Russian-language and given according to the Russian alphabetical order in the original Russian text. We keep the order of references in this translation.

[6] S. Yu. Maslov, G. E. Mints. Theory of proof search and the inverse method. Appendix to Russian translation of: C.-L. Chang, R. C.-T. Lee. Symbolic Logic and Mechanical Theorem Proving. Nauka, 1983. Pp. 291–314. (In Russian.)

[7] V. Novak, I. Perfilieva, J. Mockor. Mathematical Principles of Fuzzy Logic. Kluwer Academic Publishers, 1999.

[8] E. Gamma, R. Helm, R. Johnson, J. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.

[9] V. Chandru, M. R. Rao. Linear programming. In M. J. Atallah, ed.: Algorithms and theory of computation handbook. CRC Press, 1999.

[10] A. Ciabattoni, C. G. Fermüller, G. Metcalfe. Uniform rules and dialogue games for fuzzy logics. In F. Baader, A. Voronkov, eds.: Proceedings of LPAR 2004. Pp. 496–510.

[11] A. Degtyarev, A. Voronkov. Equality reasoning in sequent-based calculi. In A. Robinson, A. Voronkov, eds.: Handbook of Automated Reasoning. Elsevier, 2001, vol. I. Pp. 611–706.

[12] S. Gottwald. A Treatise on Many-Valued Logics. Research Studies Press, 2001.

[13] R. Hähnle. Many-valued logic and mixed integer programming. Annals of Mathematics and Artificial Intelligence, 1994, vol. 12, no. 3–4. Pp. 231–263.

[14] P. Hájek. Metamathematics of Fuzzy Logic. Kluwer Academic Publishers, 1998.

[15] P. Hájek. What is mathematical fuzzy logic. Fuzzy Sets and Systems, 2006, vol. 157, no. 5. Pp. 597–603.

[16] G. Metcalfe, N. Olivetti, D. M. Gabbay. Łukasiewicz logic: from proof systems to logic programming. Logic Journal of the IGPL, 2005, vol. 13, no. 5. Pp. 561–585.

[17] N. Olivetti. Tableaux for Łukasiewicz infinite-valued logic. // Studia Logica, 2003, vol. 73, no. 1. Pp. 81–111.

[18] L. A. Zadeh. Fuzzy logic and approximate reasoning. Synthese, 1975, vol. 30. Pp. 407–428.

The author's main papers on the thesis theme

[1′] A. S. Gerasimov. An infinite-valued predicate logic with a connective for strengthening statements. In: Proceedings of the IX All-Russian scientific conferences "Contemporary logic: problems of theory, history and applications in science" (Saint-Petersburg, 2006). Pp. 348–350. (In Russian.)

[2′] A. S. Gerasimov. A sequent calculus-based predicate logic meant for modeling of continuous scales. In: Proceedings of the Tenth national conference on artificial intelligence CAI-06 (Obninsk, 2006). Pp. 339–347. (In Russian.)

[3′] A. S. Gerasimov. A software implementation of proof search for an infinite-valued logic based on linear inequalities. In O. B. Lupanov, ed.: Proceddings of the XVI International school-seminar "Synthesis and complexity of control systems" (Saint-Petersburg, 2006). Pp. 30–35. (In Russian.)

[4′] A. S. Gerasimov. Development of an API for testing consistency of systems of linear two-term inequalities. In: Proceedings of the Contest-conference for students, PhD students, and young scientists of North-West "Microsoft technologies in theory and practice of programming" (Saint-Petersburg, 2005). Pp. 161–162. (In Russian.)

[5′] A. S. Gerasimov, N. K. Kossovski. Genuinely polynomial algorithm for determining consistency of systems of two-term linear inequalities. In D. A. Ovsyannikov, L. A. Petrosyan, eds.: Proceedings of the international conference "Stability and control processes" (Saint-Petersburg, 2005). Pp. 779–785. (In Russian.)

[6′] A. S. Gerasimov, N. K. Kossovski. Complexity evaluation of the genuinely polynomial algorithm for testing feasibility of systems of two-term linear inequalities. Bulletin of Saint-Petersburg University. Series 10, 2006, issue 2. Pp. 16–21. (In Russian.)