The Canonical Forms of Logical Formulae over the Data Types and Their Using in Programs Verification

Michael Lvov[1], Vladimir Peschanenko[1], Oleksandr Letychevskyi[2], Yuliia Tarasich[1]

[1]Kherson State University, Universitetskaja St. 27, 73000, Kherson, Ukraine
[2]Glushkov Institute of Cybernetics of NASU, Glushkov ave., St., 03187, Kyiv, Ukraine

Lvov@ksu.ks.ua, vpeschanenko@gmail.com, lit@iss.org.ua, YuTarasich@gmail.com

Abstract. A brief review and the results of working with tools for the formulae simplifying are presented. The algorithm for constructing the canonical forms of linear semi-algebraic formulae over the enumerated and multiple types is described.

Keywords: System of linear inequalities, canonical forms, logical formulae, linear semi-algebraic formulae, trapezoids.

1 Introduction

A software quality is one of the main tasks of software developers. Therefore, software developers are increasingly focusing on the verification which gives them possibilities to search errors before writing a source code of the system and also to prove various properties of models that were built on the already written code.

In general, methods of the formal models proof can be divided into static and dynamic, specific and symbolic. The main problem of static methods for proving properties of a model is the work with the fairly complex formulae, the complexity of which can be increased step by step (the construction of invariants, etc.). The main problem of dynamic methods is a well-known problem in specific modeling - the exponential explosion of the number of passed states in the model.

On the other hand, there is a lot of number of the well-researched algorithms for proving properties of models (static and dynamic methods). The problem, which arises in the modeling of specific models is that we get a big numbers of results as a result of proving their properties. The symbolic modeling is deprived of such problems. Each state in it is the certain formula that covers a set of specific states. But despite the fact that the number of states of the symbol model which is given by a formula is less than in the specific model, in the symbolic modeling the difficult questions of determining the reachability of passed states in the process of using dynamic methods and the fairly complex formulae are raised. The complexity of these formulae can grow step by step if the canonical or normal form is lacking. This implies that the

problem of construction of the canonical and normal forms is one of the most important problems in the symbol modeling.

In the first part of the article we give a brief review and the analysis of results of working with the tools for simplifying formulae. In the second part we describe the algorithm for constructing the canonical forms of linear semi-algebraic formulae off the enumerated and multiple types.

2 Tools for the Formulae Simplification

There are a lot of tools for prove. All of them can be divided into several types:

- The tools that are based on the extensions of the propositional logic or of the first-order logics. In this category, the very known tools are: ACL2, E (and E-SETHEO), KeY, Vampire, Waldmeister, Darwin.

- The tools that are based on the high order logic. The most famous and widely used of them are: PVS, HOL, Isabelle, Coq.

In this paper, we consider such systems as CVC4, MathSAT5, QEPCAD, Singular, COCOA, MiniZinc, STP, RedLog, Satallax, Isabelle, E-SETHEO, Minisat, SMTInterpol, TPS / ETPS, Paradox, Gandalf, Z3, Vampire. The main attention in the analysis of the considered systems was paid to the availability of tools for the formulae simplifying. The brief characteristics of these systems and the results of their tests are given below.

CVC4 1.4. CVC4 is an efficient open-source automatic theorem prover for satisfiability modulo theories (SMT) problems. It can be used to prove the validity (or, dually, the satisfiability) of the first-order formulae in a large number of built-in logical theories and their combinations [1]. To simplify the formulae in the prover the TRANSFORM method is used. To obtain normal results, it is necessary to put the formulae in brackets which determine a problem with the implementation of the priority of operations. Moreover, the application of the simplification method makes the formulae more cumbersome (the existence quantifier is always replaced by a generality quantifier; many symbols of negation appear; can't to remove the quantifiers in an unclosed formula). Positive characteristics of the prover – it's the ability to use it in projects that are developed under C ++, the availability of its documentation and the user-friendly syntax.

MathSAT5 5.3.10 - is an efficient SMT solver. MathSAT 5 is the successor of MathSAT 4, supporting a wide range of theories (including e.g. equality and uninterpreted functions, linear arithmetic, bit-vectors, and arrays) and functionalities (including e.g. the computation of Craig interpolants, the extraction of unsatisfiable cores, the generation of models and proofs, and the ability of working incrementally) [2, 3]. As an input formats it supports SMT-LIB 2, SMT-LIBv1.2, DIMACS. Works with GMP (Gnu Multiprecision library) and GNU C Library. Can be used in C / C ++ projects. There is a verification of SAT. The input format is limited to the libraries that it uses as the input format. It is easy to install and has good documentation, but, unfortunately, the prover doesn't have the tools for the formulae simplifying, which means that it isn't suitable for solving our problem.

QEPCAD Version B 1.69 - is an interactive command-line program written in C/C++, which is based on the SACLIB library [4]. One of the most important features of QEPCAD is its ability to produce the simple quantifier-free equivalent formulae [22], which points out the possibility of its using to simplify formulae.

This is a very good system for the formulae simplifying. In addition to the standard quantifiers it includes the existence quantifier "E" and the generality quantifier "A". Also it supports the additional quantifiers: "F" - for infinitely many; "G" - for all but certainly many; "C" - for the connected set, "xk" – for the exact number of k different values. Its shortcomings, in our opinion, is the following: 1) The QEPCAD does not understand the priority of the usage of logical operators; 2) It doesn't support the functional symbols (You need to use the algorithms for replace of functionals on the variables, that makes the formula more cumbersome); 3) It uses the different parentheses (() – for numeric operations, [] – for logical operations); 4) The negative numbers must be enclosed in the parentheses and there is no sign of multiplication (5 * x => 5x), which isn't always convenient.

Singular 4.0.3 is the computer algebra system for polynomial computations with the special emphasis on commutative and non-commutative algebra, algebraic geometry, and singularity theory. It is free and open-source under the GNU General Public Licence [5,6]. The system is easy to install, well documented and has a good syntax. There is a calculation of invariants. Can be used as the computer algebra system, but it isn't suitable for solving our problem, since it isn't developed as a system for working with the mathematical logic (quantifiers are absent, inequalities for logical formulae can't be used, there are no checks for SAT proofs).

COCOA 5.1.3. CoCoA (Computations in Commutative Algebra) [7] is the free computer algebra system to compute with numbers and polynomials. The CoCoA (CoCoALib [8]) is available under GNU General Public License. Like as Singular, the CoCoA system doesn't have the functionality for working with the mathematical logic (it does not support the quantifiers, it does not simplify the formulae), and therefore it can not be used to solve our problems.

MiniZinc (WIndows) – is the medium-level constraint modelling language. It is designed for an easy interaction with the various server solvers. In particular, MiniZinc allows the specification of global constraints by decomposition [9].

MiniZinc is written in C ++. The system works only with integer and rational numbers. It is a very good tool for modeling and programming. It is easy to install and has a good documentation. It also has an IDE which makes it easier to work with it. Unfortunately, it is not suitable for solving our problem. In the model with the help of such functions as *solve satisfy, solve maximize, solve minimize*, we set the constraints on variables (constraint programming) and then we find the solution that satisfies these constraints.

STP 2.1.2. - is the constraint solver that can solve many kinds of problems including those by program analysis tools, theorem provers, automated bug finders, cryptographic algorithms, intelligent fuzzers and model checkers [10]. Supports the CVC, SMT-LIB1, and SMT-LIB2 data input formats. It is recommended to use the SMT-LIB2 (although not all features are used in STP). Works with bit vectors and arrays. Can't work with quantifiers. There is a SAT. As stated in the description, it has a sim-

plifier that includes containing simplification algorithms, but in the sense in which we need simplifications (we gave a formula → at the output we got a simplified one) it does not work.

RedLog (Reduce _2015_10_20 Windows) – is an integral part of the interactive computer algebra system Reduce. It supplements Reduce's comprehensive collection of powerful methods from symbolic computation by supplying more than 100 functions on first-order formulae [11]. The system has a good syntax, a detailed documentation and it is simple in installation. It supports the functions, quantifiers and priority of operations. But, it simplified only two formulae. The remaining formulae were rewritten in a different form. The minuses of the system are: the absence of factorization; the absence of SAT and the proof of formulae. The system is written in LISP, so its integration with C ++ systems is complicated.

Satallax 2.8 - Satallax is the automated theorem prover for higher-order logic Satallax progressively generates higher-order formulae and corresponding propositional clauses. These formulae and propositional clauses correspond to a complete tableau calculus for higher-order logic with a choice operator. Satallax uses the SAT solver MiniSat as an engine to test the current set of propositional clauses for unsatisfiability [12]. It supports the high-order logic. As a data entry format it uses the THF language in the TPTP library. There is no documentation of the system; there is only a brief description. The program isn't oriented on the simplifying of the predicate formulae. It performs its SMT-prover tasks, proves the satisfactories or unsatisfactories of the problem, but does not simplify the formulae.

Isabelle2016 (Windows) is the generic proof assistant. It allows mathematical formulae to be expressed in a formal language and provides tools for proving those formulae in a logical calculus. The main application is the formalization of mathematical proofs and in particular *formal verification*, which includes proving the correctness of computer hardware or software and proving properties of computer languages and protocols. The most widespread instance of Isabelle nowadays is *Isabelle/HOL*, which provides a higher-order logic theorem proving environment that is ready to use for big applications [13]. It is easy to install and has a good documentation. It supports the work with the first and high order logic. It is a good system in terms of simplifying the formulae. The rules for formulae simplifying are defined by users. In this regard, it works as a system of rewriting rules. Therefore, the responsibility of simplifications rests with the user. Written in SML, so the integration of this system with C ++ is very difficult.

E-SETHEO. E-SETHEO is the strategy-parallel compositional theorem prover for the first-order logic with equality. It combines a variety of high-performance theorem provers and specialized decision procedures into one of the most powerful ATP systems currently available. The core idea of the E-SETHEO framework is to let different proof search procedures compete for resources for solving the given problem [14].

Unfortunately, we can't install this system. The main problem is that the project was abandoned for many years, and besides this the system is written on many programming languages such as C, Eclipse Prolog, Perl, Bigloo Scheme.

Minisat 2.2.0 – is the minimalistic, open-source SAT solver [15]. The solver is written in C ++. It has MIT license. It accepts the input data in DIMACS format (CNF - data input format). There is no simplification of the formulae; therefore the data input format should be in the conjunctive normal form. Therefore, it can't be used to solve our problem.

SMTInterpol 2.1. SMTInterpol is the SMT Solver that can compute Craig interpolants for various theories. The current version of SMTInterpol supports interpolation for the combination of the quantifier-free fragments of the theories of uninterpreted functions, linear real arithmetic, linear integer arithmetic, and arrays [16]. It uses a SMT-LIB as the input data format. It is written in Java and can be adapted for C ++. It has an experimental tool for the formulae simplifying. It does not work with quantifiers. As a result it could not simplify any of the proposed formulae.

TPS/ETPS. TPS and ETPS are, respectively, the Theorem Proving System and the Educational Theorem Proving System [17]. Unfortunately, there is no access to download the system, and the available sources do not work. The system is written in Common Lisp, so their integration into the C ++ system is complicated.

Paradox 3.0. – Paradox is the automated theorem proving system developed by Koen Lindström Claessen and Niklas Sörensson at the Chalmers University of Technology. The software is written in the Haskell programming language and is released under the terms of the GNU General Public License and is free [18]. It is only in the public folder of the provers of previous CASC tests. It doesn't work like the previous systems.

Gandalf c-2.6.r1 - is the first-order automated theorem prover applied to several domain-specific tasks such as Semantic web. It is programmed in the Scheme programming language which is then compiled to the C programming language using Hobbit from SCM [19]. The access to the prover and its documentation is closed to those who do not have an access to the account of the University of Tallinn. The last design date of the work over this prover was the 2003. Since the Gandalf did not take part in the competitions recent years and we did not get any answers to the letters to the authors, we can assume that its development stopped.

Vampire 2.6 (Vampire 4.0) is the automatic theorem prover for the first-order logic. It was used in a number of academic and industrial projects [20]. Vampire implements the calculi of the ordered binary resolution and the superposition for the handling equality. Internally, Vampire works only with a clausal normal form. Problems are classified during preprocessing. Vampire implements many useful preprocessing transformations including the Sine axiom selection algorithm [21]. It is said in the official website that it will be possible to download the prover, after the release of the 3rd version (although the 4.0 version is already participating in the CASC). The previous versions cannot be downloaded also. Source codes with CASC contain errors. So, the prover could not be tested.

Z3 is a state-of-the art theorem prover from Microsoft Research. It can be used to check the satisfiability of logical formulae over one or more theories. Z3 offers a compelling match for software analysis and verification tools, since several common software constructs map directly into supported theories [26]. Z3 is a low level tool. It is best used as a component in the context of other tools that require solving logical

formulae. Consequently, Z3 exposes a number of API facilities to make it convenient for tools to map into Z3, but there are no stand-alone editors or user-centric facilities for interacting with Z3. The language syntax used in the front ends favor simplicity in contrast to linguistic convenience [41, 42].

As it can be seen from the material described above, not all considered systems were launched and analyzed. The Isabelle2016 is only one system that can completely done our task. But, in turn, as was mentioned above, it doesn't contain a set of rules for simplification (the rule process is set by the user himself).

As for QEPCAD ver. B 1.69 then from the point of view of simplifying of the formulae – the system works well, but it has a fairly complex and cumbersome syntax and does not have a good documentation.

The most used languages for creating such systems are C / C ++ and ML-languages (OCaml, SML).

In all systems, we are faced with the problem of the need for rewriting the formulae in a certain format. It is very difficult to rewrite formulae and bring them to the required format, especially if parentheses are to be placed everywhere, such as in SMT-LIB. Accordingly, the development of systems for the formulae simplifying, which would allow the posed tasks is an open problem. In addition, no less important is the need to implement appropriate algorithms that make it possible to simplify formulae at the more efficient level.

3 The Canonical Forms of Linear Semialgebraic Formulae

Introduce the basic notation:

$X = \{x_1,...,x_n\}$ - is a n - dimensional vector of variables.

Q - is a constructive linearly ordered field, called as the coefficient field.

Q^n - is a vector space over Q.

$\bar{a} = (a_1,...,a_n) \in Q^n$ - is a vector of numerical coefficients,

$b \in Q$ - is a numerical coefficient.

$LF(\bar{a}, X)$ -is a linear form from $X, \bar{a}$: $LF(a, X) \overset{df}{=} a_1x_1 + ...+a_nx_n$

The linear inequality (LI) $E(\bar{a}, X, c)$ has a form $LF(\bar{a}, X) \le c$ b and is denoted by $E(\bar{a}, X, c)$. To shorten the record, the linear inequalities will be written in the form $E(X)$, indicating only the variables on which it depends.

Definition 1. A linear semialgebraic formula (LSF) over Q from X is a quantifier-free logical formula $F(X) = F(E_1(\bar{a}_1, X, c_1),...,E_m(\bar{a}_m, X, c_m))$ of linear inequalities $E_1(X),..E_m(X)$ in the signature of logical connectives $\&,\vee$. The linear semialgebraic set $M(F)$ is a set $M(F)) = \{V \in Q^n \mid F(V)\}$.

Definition 2. Let x - be a variable that is on the left-hand side of LI with nonzero coefficient and $Y = X - \{x\}$. Then the LI can be transformed to the form

$$x \le LF(\bar{b}, Y) + c \quad \text{or} \quad x \ge LF(\bar{b}, Y) + c \,. \tag{1}$$

Such form of LI will be called resolved with respect to or -resolved. If each LI of a linear semi-algebraic formula is depending on x, is represented in the allowed form, then the LSF will be called represented in the x-allowed form. Is represented in the allowed form, then the LSF will be called represented in the x -allowed form.

By transforming the formula $F(E_1,...,E_m)$ according to the rules of propositional algebra of a disjunctive normal form, one can obtain its representation in the form of a set of LI systems. However, such representation is not the only one and is therefore poorly suitable for computer algebra algorithms. The main result of this work is the definition of the canonical form of a LSF that is possessing by the property of uniqueness and by the other useful properties, and the description of its construction algorithm. The proposed canonical form of a LSF is the direct generalization of the canonical form of the LI system representation and the algorithm for its construction, those where presented in [29, 30]. The algebraic programming methodology used in [29] lies in a constructive definition of a multisorted algebraic system [31] whose objects are represented in the form of expressions, and algorithms compute values of these expressions, i.e., construct their canonical forms [32–37]. The basic canonical form of an SLI is the polyhedron of solutions to the SLI in the form of the union of a finite number of trapezoids. This canonical form uses the idea of projection (elimination of a quantifier) from the Fourier-Motzkin [38,39] and Tchernikov [40] methods.

Definition3. The formula of a x -segment $s = [L(Y), x, R(Y)]$ over a set of variables $Y, x \notin Y$, is a double linear inequality in the form. $L(Y) \le x \le R(Y)$.

If $Y = \{y_1,....,y_m\}$, then the formula $s = [L(Y), x, R(Y)]$ is interpreted on the space Q^{m+1} as an elementary dihedral area $M(s) = \left\{(q_0, \bar{q}) \in Q^{m+1} \mid L(\bar{q}) \le q_0 \le R(\bar{q})]\right\}$ - x - segment.

Inequalities that are represented in the x -resolved form of the form $x \le R(Y)$ or $x \ge L(Y)$, can be transformed in x -segments: $s = [-\infty, x, R(Y)]$ or $s = [L(Y), x, +\infty]$.

Let $M \subset Q^m$ and $s = [L(Y), x, R(Y)]$. Define the restriction $s \cdot M$ of segment s on M as follows:: $s \cdot M = [L(Y), x, R(Y)] \cdot M \overset{df}{\approx} (L(Y) \le x \le R(Y)) \,\&\, (Y \in M)$. $s \cdot M \ne \varnothing$ if and only if the condition is satisfied $\forall \bar{q}(\bar{q} \in M) \to (L(\bar{q}) \le R(\bar{q}))$. This condition will be written in the form $L(x) \le_M R(x)$. It is obvious that there is a property

$$M \cdot [L, x, R] \cdot M' = M \cdot [L, x, H] \cdot M' + + M \cdot [H, x, R] \cdot M' \,. \tag{2}$$

The relation (2) defined on the set of segments the *operation of reduction* and the inverse *operation of partitioning* the set on the part. The exact definition of the partition is given below.

2.1. Systems of Linear Inequalities and Trapezoids

On the vector variables $X = \{x_1,...,x_n\}$ defined the order $x_1 < ... < x_n$ and denoted $X_j \overset{df}{=} \{x_j,...,x_n\}$

Let $s_j = [L_j(X_{j-1}), x_j, R_j(X_{j-1})], j = 1,...,n-1$,

$s_n = [L_n, x_n, R_n], L_n, R_n \in Q$.

Definition 4. The set T, defined by the formula $T = s_1 \cdot s_2 \cdot ... \cdot s_n$ (the parentheses are grouped by default to the right), is called a trapezoid in space Q^n.

Denote by T_j the trapezoid $s_j \cdot s_{j+1} \cdot ... \cdot s_n$. Then

$T = T_1$, $T_j = s_j \cdot T_{j+1}, j = 1,2,...,n-1$,

And T_{j+1} - is the projection T_j on the subspace Q^{n-j+1}, that are determined by the coordinates $\{x_{j+1},...,x_n\}$.

Thus, the trapezoid T is defined by a sequence of projections onto a decreasing sequence of subspaces $Q^n \supset Q^{n-1} \supset ... \supset Q^1$, and each projection is also a trapezoid.

The representation of a trapezoid T in the form of a sequence of its projections will be denoted by the formula

$T = T^{(n)} \to T^{(n-1)} \to ... \to T^{(1)}$.

A convex set $P \subseteq Q^n$, that is representing the solution of a system of linear inequalities is called a polygon.

Definition 5. By a partition of a set $A \subset Q^n$ we mean its representation in the form of a union of subsets $A = A_1 + + A_2 + + ... + + A_k$, that $i \neq j \to Dim(A_i \cap A_j) < n$.

This means that different elements of the partition cross over a set whose dimension is less than the dimension of the space (only a common boundary can exist).

The polygon P can be represented in the form of a partition $P = T_1 + + ... + + T_m$ into trapezoids $T,...,T_m$**.** The algorithm for this representation is described in [29].

Definition 6. The partition $P = T_1 + + ... + + T_m$ will be called irreducible if for any pair of trapezoids T_i, T_j can't be applied the operation (2).

Arrangement of elements of polygon partitioning on trapezoid. The irreducible partition of the polygon into trapezoids is unique up to permutations of the trapezoids of the partition.

The sequence can be ordered in accordance with the following properties. If $s_1^{(n)},...,s_m^{(n)}$ then the sequence of projections of an irreducible partition of a polygon $P = T_1 + + ... + + T_m$ onto the axis Ox_n, then or $s_i^{(n)} = s_j^{(n)}$, or $Dim(s_i^{(n)} \cap s_j^{(n)}) = 0$, i.e. $s_i^{(n)}, s_j^{(n)}$ either coincide, or don't intersect, or are adjacent. Therefore, $s_1^{(n)},...,s_m^{(n)}$

can be represented so that they are located on the axis Ox_n in ascending order. If $s_{i_1}^{(n)},...,s_{i_l}^{(n)}$ are pairwise distinct numerical segments, then the ordering will have the form

$$s_1^{(n)} = ... = s_{i_1}^{(n)} = [L_1, x_n, L_2],\ s_{i_1+1}^{(n)} = ... = s_{i_2}^{(n)} = [L_2, x_n, L_3],..., s_{i_{l-1}+1}^{(n)} = ...$$

$$= s_m^{(n)} = [L_l, x_n, L_{l+1}],$$

where $L_1 < L_2 < ... < L_{l+1}$. Then the polygon P can be represented in the form

$$P = (T_1 + +...+ + T_{i_1}) + + (T_{i_1+1} + +...+ + T_{i_2}) + +...+ + (T_{i_l+1} + +...+ + T_m)$$

or

$$P = (T_1^{(n-1)} + +...+ + T_{i_1}^{(n-1)}) \cdot s_{i_1}^{(n)} + + (T_{i_1+1}^{(n-1)} + +...+ + T_{i_2}^{(n-1)}) \cdot s_{i_2}^{(n)} + +...$$

$$+ + (T_{i_l+1}^{(n-1)} + +...+ + T_m^{(n-1)}) \cdot s_m^{(n)}.$$

Denote the partitions of trapezoids in parentheses through $P_{i_1}^{(n-1)}, P_{i_2}^{(n-1)},...,P_m^{(n-1)}$.

Then $P = P_{i_1}^{(n-1)} \cdot s_{i_1}^{(n)} + + P_{i_2}^{(n-1)} \cdot s_{i_2}^{(n)} + +...+ + P_m^{(n-1)} \cdot s_m^{(n)}$.

The property of polygon partitioning that are described above holds for projections of any dimension. Namely, let $P = T_1 + +...+ T_l$ - is a irreducible partition of a polygon with common $k+1$ - th projection $T^{(k+1)}$:

$$T_1 = T_1^{(1)} \to ... \to T_1^{(k)} \to T^{(k+1)},...,T_l = T_l^{(1)} \to ... \to T_l^{(k)} \to T^{(k+1)}.$$

Then $T,...,T_l$ can be ordered in such a way that the *canonicity property* is fulfilled:

$$s_1^{(k)} = ... = s_{i_1}^{(k)} = [L_1, x_k, L_2],\ s_{i_1+1}^{(k)} = ... = s_{i_2}^{(k)} = [L_2, x_k, L_3],..., s_{i_{l-1}+1}^{(k)} = ...$$

$$= s_m^{(k)} = [L_l, x_k, L_{l+1}], \quad (2.1)$$

$$L_1 <_{T^{(k+1}} L_2 <_{T^{(k+1}} ... <_{T^{(k+1}} L_{l+1}. \quad (2.2)$$

An irreducible partition of the polygon into trapezoids ordered in accordance with the canonical property will be called the canonical decomposition or the canonical sum.

Since $T_i^{(j)}$ = projection $T_i^{(j+1)}$ onto a subspace Q^j, the Definition 5 means that in the canonical sum any projections of trapezoid summands can intersect only along their boundary. Relation (2) can be called an operation of bringing of similar.

Theorem 1. Let be a system of linear inequalities and $P \subset Q^n$ ia a polygon (a convex set in Q^n), defined by this system. Then P can be represented in the form of the canonical sum of trapezoids $P = T_1 + +...+ + T_k$.

Algorithm 1 (intersection of trapezoids). On the set of trapezoids in space Q^n it is natural to define the operation of intersection $T_1 \cap T_2$. As shown in [29], this intersection can be represented as the canonical sum of trapezoids. Consider the algorithm

for computing the intersection $T_1 \cap T_2 = T_1^{'} + + \ldots + + T_k^{'}$. Let T_1, T_2 in the form $T_1 = T_1^{(n-1)} \cdot s_{1n}$, $T_2 = T_2^{(n-1)} \cdot s_{2n}$. Then $T_1 \cap T_2$ is evaluated recursively "from below-up".

Recursion basis:

$$s_{1n} \cap s_{2n} = \varnothing \rightarrow T_1 \cap T_2 = \varnothing \quad else \quad T_1 \cap T_2 = (T_1^{(n-1)} \cap T_2^{(n-1)}) \cdot (s_{1n} \cap s_{2n}),$$
$$\max(L_{1n}, L_{2n}) \leq_T \min(R_{1n}, R_{2n})] \rightarrow s_{1n} \cap s_{2n} = [\max(L_{1n}, L_{2n}), x_1,$$
$$\min(R_{1n}, R_{2n})] else\, s_{1n} \cap s_{2n} = \varnothing \qquad (3)$$

Recursion step:

Suppose that $T_1 = T_1^{(n-k)} \cdot T_1^{(k)}$, $T_2 = T_2^{(n-k)} \cdot T_2^{(k)}$ and $T_1^{(k)} \cap T_2^{(k)}$ has already been calculated, and $T_1^{(k)} \cap T_2^{(k)} = T_1^{'} + + \ldots + + T_l^{'}$. Then, due to the distributive mark «++»with respect to a mark «·», the computation reduces to the calculation $(T_1^{(n-k)} \cap T_2^{(n-k)}) \cdot T_j^{'}$, and this calculation, in turn, reduces to calculation $(s_{1n-k} \cap s_{2n-k}) \cdot T_j^{'}, j = 1, \ldots, l.$ The result of the operation must satisfy the conditions in (3).

Since $\max(L_1, L_2), \min(R_1, R_2)$ are not linear combinations, the result of the operation of intersection of segments $s_{1n-k} \cap s_{2n-k}$ on $T_j^{'}$ depends on the mutual arrangement of these segments on the trapezoid $T_j^{'}$.

The relation $L(Y) \leq_T R(Y)$ is defined as follows: $L(Y) \leq_T R(Y)$, if for any vector $Y_0 \in T$ the inequality $L(Y_0) \leq R_k(Y_0)$ holds. In the canonical form of a representation of a trapezoid T, these relations must be satisfied for any value $k = n, n-1, \ldots, 1$ on subspaces Q^k.

Let us first consider the problem of calculating the intersection $s_1 \cap s_2$ on T in general terms.

Let $s_1 = [L_1, x, R_1], s_2 = [L_2, x, R_2]$, $T_1 = s_1 \cdot T, T_2 = s_2 \cdot T$, and $L_1 \leq_T R_1$, $L_2 \leq_T R_2$.

$$\text{Let } LL_{ij} = T \cap (L_i \leq_T L_j), \quad RR_{ij} = T \cap (R_i \leq_T R_j), \quad LR_{ij} \overset{df}{=} L_i \leq_T R_j,$$
$$[LR]_{ij} \overset{df}{=} [L_i, x, R_j], \; i, j = 1,2 \qquad (4)$$

And we list the relations in the form of rewriting rules for all variants of the mutual arrangement of the lower and upper caps T_1, T_2, writing formulae (3) in terms of (4):

$RR_{12} \rightarrow$ (// **variants 1-6**

1. $LR_{12} \rightarrow T_1 \cap T_2 = \varnothing$,
2. $(L_2 \cap R_1 \neq_T \varnothing) \,\&\, LL_{12} \rightarrow T_1 \cap T_2 = [LR]_{21} \cdot LR_{21}$,

3. $(L_2 \cap R_1 \neq_T \varnothing) \& (L_2 \cap L_1 \neq_T \varnothing) \to T_1 \cap T_2 = [LR]_{21} \cdot LL_{21} \& LR_{21} + + [LR]_{11} \cdot LL_{21}$

4. $LR_{21} \& LL_{12} \to T_1 \cap T_2 = [LR]_{21} \cdot T$,

5. $LL_{21} \& (L_1 \cap L_2 \neq_T \varnothing) \to T_1 \cap T_2 = [LR]_{21} \cdot LL_{12} + + [LR]_{11} \cdot LL_{21}$,

6. $LL_{12} \to T_1 \cap T_2 = T_2$

);

$(R_2 \cap R_1 \neq_T \varnothing) \& LR_{12} \to ($ // **variants 7-11**

1. $(L_2 \cap R_1 \neq_T \varnothing) \& LL_{12} \to T_1 \cap T_2 = [LR]_{21} \cdot LR_{21} \& RR_{12} + + [LR]_{22} \cdot RR_{21}$

2. $(L_2 \cap R_1 \neq_T \varnothing) \& (L_2 \cap_T L_1 \neq \varnothing) \to T_1 \cap T_2 = [LR]_{21} \cdot LL_{12} \& LR_{21} + + [LR]_{11} \cdot RR_{12} \& LL_{21} + + [LR]_{12} \cdot RR_{21}$,

3. $LR_{21} \& LL_{12} \to T_1 \cap T_2 = [LR]_{21} \cdot RR_{21} + + [LR]_{22} \cdot RR_{12}$,

4. $LR_{21} \& (L_1 \cap_T L_2) \neq \varnothing) \to T_1 \cap T_2 = [LR]_{21} \cdot LL_{12} + + [LR]_{11} \cdot LL_{21} \& RR_{12} + + [LR]_{12} \cdot RR_{21} \& LL_{21}$ 5.

$LL_{21} \to T_1 \cap T_2 = [LR]_{11} \cdot RR_{12} + + [LR]_{12} \cdot RR_{21}$

);

$(R_2 \cap R_1 \neq_T \varnothing) \& (R_2 \cap L_1 \neq_T \varnothing) \to ($ // **variants 12-14**

1. $(R_1 \cap L_2 \neq_T \varnothing) \& (L_1 \cap L_2 \neq_T \varnothing) \to T_1 \cap T_2 = [LR]_{21} \cdot RR_{12} \& LL_{12}, + + [LR]_{22} \cdot RR_{12} \& LL_{21} + + [LR]_{12} \cdot RR_{21} \& LL_{21},$

2. $LR_{21} \& (L_1 \cap L_2 \neq_T \varnothing) \to T_1 \cap T_2 = [LR]_{21} \cdot RR_{12} \& L_{12} + + [LR]_{11} \cdot RR_{12} \& LL_{21} + + [LR]_{12} \cdot RR_{21} \& LL_{21},$

3. $LL_{21} \to T_1 \cap T_2 = [LR]_{12} \cdot RR_{21}$

);

$RR_{21} \& LR_{12} \to ($ // **variants 15-17**

1. $LL_{12} \to T_1 \cap T_2 = T_2$,

2. $L_1 \cap L_2 \neq_T \varnothing \to T_1 \cap T_2 = [LR]_{22} \cdot LL_{12} + + [LR]_{12} \cdot LL_{21}$,

3. $LL_{21} \to T_1 \cap T_2 = [LR]_{12} \cdot T$

);

$RR_{21} \& (L_1 \cap R_2) \neq \varnothing \to ($ // **variants** 18-19

1. $L_1 \cap L_2 \neq_T \varnothing \to T_1 \cap T_2 = [LR]_{22} \cdot LL_{12} + + [LR]_{12} \cdot LL_{21}$,

2. $LL_{21} \to T_1 \cap T_2 = [LR]_{12} \cdot LR_{21}$

);

1. $RR_{21} \to T_1 \cap T_2 = \varnothing$. // **variant** 20

Algorithm 2 (Variant recognition) is described in [30].

Algorithm 3 (Intersection of polygons). The operation of polygons intersection $P_1 \bigcap P_2$ is described by a system of ratio - rewriting rules. Let

$P_1 = T_{11} + +T_{12} + +...+ +T_{1k_1}$, $P_2 = T_{21} + +T_{22} + +...+ +T_{2k_2}$.

Represented P_1, P_2 as $P_1 = g + G$, $P_2 = h + H$, where

$g = T_{11}$, $h = T_{21}$, $G = T_{12} + +...+ +T_{1k_1}$, $H = T_{22} + +...+ +T_{2k_2}$.

The basic rule is:

$$(g + +G)(h + +H) = gh + +(gH \cup hG) + +HG \quad (5)$$

Let $g = T' \cdot [l_g, r_g]$, $h = T'' \cdot [l_h, r_h]$. Then, if $r_g \le r_h$ $gH = \varnothing$, and if $r_h \le r_g$ $hG = \varnothing$. This makes it possible to simplify rule (5). We introduce the following access functions: for $T = T' \cdot s$, $Base(T) \overset{df}{=} s$, for $s = [L, x, R]$ $LP(s) = L$, $RP(s) = R$. Then (4) simplify:

$$RP(Base(g)) \le RP(Base(h)) \to (g + +G)(h + +H) = gh + +hG + +HG$$
$$else\ (g + +G)(h + +H) = gh + +gH + +HG \quad (6)$$

Derivatives rules are derived from (5) for special cases $G = \varnothing$ or $H = \varnothing$:

$$RP(Base(g)) \le RP(Base(g)) \to g(h + +H) = gh\ else\ g(h + +H) = = gh + +gH \quad (7)$$
$$RP(Base(g)) \le RP(Base(h)) \to (g + +G)h = gh + +hG$$
$$else\ (g + +G)h = gh \quad (8)$$

The rewriting system (6) - (8) represents an algorithm for polygons intersecting.

2.2. Linear Semi-Algebraic Formulae and Polytrapezoids

Definition6. Let $Y = (y_1, ..., y_m)$, $x \notin Y$, $T \subseteq Q^m$. The algebra of x - polysegments (polysegments) over the base T is a constructive extension of the algebra of x - segments which elements of carrier have a form: $S \cdot T = ([L_1, x, R_1] + +[L_2, x, R_2] + +...+ +[L_k, x, R_k]) \cdot T$, $L_i =$, $= L_i(Y), R_i = R_i(Y)$ (9)

with the following conditions:

1. $L_1 \le_T R_1 \le_T L_2 \le_M R_2 \le_T ... \le_T L_k \le_T R_k$, $R_i \ne L_{i+1}$; (10)
2. if for some i $R_i \equiv L_{i+1}$, the record (3) is reduced in accordance with (2): $[L_i, x, R_i] + +[L_{i+1}, x, R_{i+1}] = [L_i, x, R_{i+1}]$.

On the algebra of polysegments defined set-theoretic operations of intersection and union. It's obvious that

$([L_1, x, R_1] + +...+ +[L_k, x, R_k]) \cdot T = [L_1, x, R_1] \cdot T \cup ... \cup [L_k, x, R_k] \cdot T$.

Definition7. Let $X_j = (x_1, ..., x_j)$, $j = 1, 2, ..., n$. Define the sequence S_j recursively:

1. $s_{1i} = [a_i, x_1, b_i], a_j, b_j \in Q, i = 1, ..., k_1$. $S_1 = s_{11} + +...+ +s_{1k_1}$ - Basic, x_1- polysergment, $T_1 \overset{df}{=} S_1$.

2. For any $j=1,...,n-1$ S_{j+1}- $x_{j+1}$is a polysergment over T_j, and $T_{j+1} \overset{df}{=} S_{j+1} \cdot T_j$.

A set T_n (or simply T) is called a polytrapecoid in space Q^n. Obviously, $T = S_n \cdot S_{n-1} \cdot ... \cdot S_1$, and T_j - projection T_{j+1} onto the subspace Q^j, defined by the coordinates $x_1,...,x_j$.

The polytrapecoid T is defined by a sequence of projections onto a decreasing sequence of subspaces $Q^n \supset Q^{n-1} \supset ... \supset Q^1$, with each projection also a polytrapezoid.

Algorithm 4 (intersection of polytrapecoids). Let PT_1, PT_2 be a polytrapecoid and $P = PT_1 \cap PT_2$. Obviously, P can be represented as a canonical sum of polytrapecoids. The algorithm for intersecting polythracoids computes P recursively "bottom-up".

Basis of recursion.

Let $PT_1 = S_{11} \cdot ... \cdot S_{1n}$, $PT_2 = S_{21} \cdot ... \cdot S_{2n}$. Then

$$PT_1 \cap PT_2 = (PT_1^{(n-1)} \cap PT_2^{(n-1)}) \cdot (S_{1n} \cap S_{2n}) \tag{11}$$

The relation (11), in accordance with Definition 6, 1 defines a recursion basis -the calculation algorithm of $S_{1n} \cap S_{2n}$, which is defined by the rewriting rules (6) - (8) and represents the sum of the segments of the numerical axis Ox_x, satisfying (10).

Step of recursion.

Let for some k $PT_1 = S_{11} \cdot ... \cdot S_{1n-k} \cdot ST^{(k)}$, $PT_2 = S_{21} \cdot ... \cdot S_{2n-k} \cdot ST^{(k)}$, , where $ST^{(k)}$ is the sum of polythracoids, representing $(S_{1n-k} \cdot ...S_{1n}) \cap (S_{2n-k} \cdot ...S_{2n})$, i.e.

$ST^{(k)} = (S_{1n-k} \cdot ...S_{1n}) \cap (S_{2n-k} \cdot ...S_{2n})$,

$$PT_1 \cap PT_2 = ((S_{11} \cdot ... \cdot S_{1n-k}) \cap (S_{21} \cdot ... \cdot S_{2n-k})) \cdot ST^{(k)} \tag{12}$$

Then we rewrite (12) in the form

$PT_1 \cap PT_2 = ((S_{11} \cdot ... \cdot S_{1n-k-1}) \cap (S_{21} \cdot ... \cdot S_{2n-k-1}))(S_{1n-k} \cap S_{2n-k}) \cdot ST^{(k)}$,

reducing the problem to the step of recursion - computing $(S_{1n-k} \cap S_{2n-k}) \cdot ST^{(k)}$.

Since $ST^{(k)}$ is the sum of polytrapecoids, $ST^{(k)} = PT_1^{(k)} + ... + PT_l^{(k)}$

$(S_{1n-k} \cap S_{2n-k}) \cdot ST^{(k)} = (S_{1n-k} \cap S_{2n-k}) \cdot PT_1^{(k)} + ... + (S_{1n-k} \cap S_{2n-k}) \cdot PT_l^{(k)}$ For the calculation, $(S_{1n-k} \cap S_{2n-k}) \cdot PT_j^{(k)}$ algorithms 1-3 described above are applied.

Algorithm 5 (unification of polythracoids). Let PT_1, PT_2 are the polythrapezoids and $P = PT_1 \cup PT_2$. P can be represented in the form of a canonical sum of polytrapecoids. Let

$PT_1 = PT_1^{(n-1)} \cdot S_{1n}$, $PT_2 = PT_2^{(n-1)} \cdot S_{2n}$.

Let $S_{12n} = S_{1n} \cap S_{2n}, S_{1n}^{'} = S_{1n} - S_{12n}$, $S_{2n}^{'} = S_{2n} - S_{12n}$

Then $S_{1n} = S_{1n}^{'} + S_{12n}$, $S_{2n} = S_{2n}^{'} + S_{12n}$ and

$PT_1 \cup PT_2 = Ord(PT_1^{(n-1)} \cdot S_{1n}^{'} + + PT_2^{(n-1)} \cdot S_{2n}^{1} + + (PT_1^{(n-1)} \cup PT_2^{(n-1)}) \cdot S_{12n}$ (13)

where the function $Ord()$ orders the summands. The calculation algorithm $S_{1n}^{'}, S_{2n}^{'}, S_{12n}$ in (13) computes S_{12n}, using algorithms 1-3 and recursively computes $PT_1^{(n-1)} \cup PT_2^{(n-1)}$. The calculation algorithm $S_{1n}^{'} = S_{1n} - S_{12n}$, $S_{2n}^{'} = S_{2n} - S_{12n}$ is similar to algorithm (3).

In conclusion, we note that the algebra of linear semialgebraic sets is constructed as a chain of extensions of the algebras of trapezoids, polygons and polytrapezoids by an overload of set-theoretic operations $\&, \vee$

Theorem 2. Every linear semialgebraic set M (see Definition1) can be uniquely represented as a sum of polytrapecoids: $M = GT_1 + + GT_2 + + ... + + GT_k$

4 The Canonical Forms of Logical Formulae off the Enumerated Type

An *enumerated data type* we have defined as a finite ordered set $A = \{a_1, a_2, ..., a_m\}$ with logical operations that are order predicates and equality predicates. We restrict the consideration with the predicate of non-strict order, because the strict order obviously is expressed by non-strict order, order is denoted as $"\leq, \geq"$ and equality is denoted as $"="$.

Consider unquantified formulae of applied logic of predicate over the type Examine unquantified formulae of applied logic of predicate off the type A in the signature of logical connectives $< \&, \vee >$ to determine their canonical forms that are analogous to the canonical forms of linear semi-algebraic formulae which were mentioned above.

The algorithm for the canonical form calculation of the logical formulae off the enumeration type

Suppose that $X = \{x_1, ..., x_n\}$, $x_1 \prec ... \prec x_n$ - is an ordered set of object variables and $F(P_1, ..., P_k; X)$ is a formula of the applied predicate logic off A by the atomic predicates $P_1, .. P_k$ and by the object variables from the set X. The atomic predicates $P_1, .. P_k$ have the form $x = y$, $x \leq y$, $x \geq y$, where x, y are/or the object variables with the aim of defining A, or the elements A.

The preliminary simplification $F(P_1, ..., P_k; X)$ is the elimination of the equalities and the bringing of atomic equalities to the form $x \leq y$ or $x \geq y$, where $x \in X$, $y \in X \cup A$, and if $y \in X$, then $x \prec y$.

Submit the x - segments $[L, x, R]$ as double inequalities $L \leq x \leq R$, and trapezoids in a form $T = s_1 \cdot ... \cdot s_n$, $s_i = [L_i, x_i, R_i]$. Each inequality can be represented as a trapezoid. Stand for $\min = a_1, \max = a_m$. Then, by convention of an enumerated type, for $x \in X$ *a priori* we have the representation

$$s_x^{(0)} = [\min, x, \max], \quad (*)$$

and for inequalities $x \le y, x \ge y$ = accordingly

$$s_{x \le y} = [\min, x, y],\ s_{x \ge y} = [y, x, \max]. \qquad (**)$$

In accordance with this, each atomic inequality $P_j = x_{j_i} \le y$ or $P_j = x_{j_i} \ge y$ of the formula $F(P_1,...,P_k; X)$ can be presented in the form of an atomic trapezoid $T^{(j)} = s_{1j} \cdot ... \cdot s_{nj}$, x_{j_i} - the segment of which has the form (**), and the other segments have the form (*). We replace the logic ligaments $\&, \vee$ by operations $\cap, \cup$. As a result, the calculation of the canonical form $F(P_1,...,P_k; X)$ is reduced to the calculation of values $F(T^{(1)},...,T^{(k)}; X)$.

Algorithms for computing of the canonical forms of trapezoid intersections, polygons, polytrapezoids and of linear semialgebraic formulae off an enumerated type are essentially represented by the foregoing algorithms. Point out, that due to the simple of right part of inequalities the algorithm 2 of the option recognition has simpler form than in the general case.

5 The Canonical Forms of Logical Formulae off the Multiple Type

Multiple data type $Set(U)$ is defined as the algebra of subsets off a finite set – universum $U = \{u_1, u_2, ..., u_m\}$, signature set-theoretic operations $< \cap, \cup, - >$ and affiliations operations $a \in A$, equality $a = b$ and negation of equality $a \ne b$. Consider the unquantified formulae of applied predicate logic off the type $Set(U)$ of variables $X = (x_1,...,x_n)$ of type Var in the signature of the logical connectives $< \&, \vee, \neg >$ to determine their canonical forms, which are analogous to canonical forms of linear semi-algebraic formulae, which were mentioned above.

First of all, give consideration the logical product of the form $\Phi = \Phi_0 \,\&\, \Phi_= \,\&\, \Phi_{\ne}$, where

$$\Phi_0 = \begin{cases} x_1 \in A_1 \\ x_2 \in A_2 \\ . \\ x_n \in A_n \end{cases},\ \Phi_= = \begin{cases} x_{i1} = y_1 \\ x_{i2} = y_2 \\ . \\ x_{ik} = y_k \end{cases},\ \Phi_{\ne} = \begin{cases} x_{j1} \ne z_1 \\ x_{j2} \ne z_2 \\ . \\ x_{jl} \ne z_l \end{cases} \qquad (1)$$

elementary conjunctions of which have the appropriate form $x \in A,\ x = y,\ x \ne z$, where $A \subseteq U,\ x, y, z \in X$. It can be assumed that the conditions are met in (1):

1. In the part Φ_0 all conjuncts are included in the form $x_j = A_j,\ j = 1,..,n$. (In particular, the conjunct can be in the form $x_j \in U$ or $x = a$, i.e. $x \in \{a\}$.

2. If in $\Phi_=$ is a conjunct in the form $x = y$, then the conjunct of the form $x \ne y$ will be absent in $\Phi_{\ne}$ (otherwise, a conjunction is false).

The following relation can be used for the simplification of (1):

$$(x \in A) \& (y \in B) \& (x = y) = (x \in A \cap B) \& (x = y) \tag{2}$$

After this transformation we obtain the conjunction of the form $\Phi = \Phi_0 \& \Phi_= \& \Phi_{\neq}$

$$\Phi_0 = \begin{cases} x_1 \in A_1 \\ x_2 \in A_2 \\ . \\ x_k \in A_k \end{cases}, \Phi_= = \begin{cases} x_{k+1} = y_1 \\ x_{k+2} = y_2 \\ . \\ x_n = y_{n-k} \end{cases}, \Phi_{\neq} = \begin{cases} x_{j1} \neq z_1 \\ x_{j2} \neq z_2 \\ . \\ x_{jl} \neq z_l \end{cases}, \tag{3}$$

where $\{y_1,...,y_{n-k}\} \subseteq \{x_1,..x_k\}$. Analyze the $X_0 = (x_1,...,x_k)$, $X_= = (x_{k+1},...,x_n)$. So, if we consider only the parts $\Phi_0, \Phi_=$ of the (3) conjunction, variables of the subset X_0 will be considered as independent, and the variables $X_=$ as correlate relations of equalities of the part $\Phi_=$ of the formula (3). Thus, we have received the canonical form $\Phi_0 \& \Phi_=$ of formula (1) with up to the order of elementary conjuncts. The uniqueness is ensured by the ordering of variables in the left part of the elementary conjuncts: $x_1 \succ x_2 \succ ... \succ x_k$; $x_{k+1} \succ x_{k+2} \succ ... \succ x_n$.

To simplify $\Phi_0 \& \Phi_{\neq}$ the formula (3) define a mapping $\varphi: X \to X_{base}$:

$$\varphi(x) = \begin{cases} x, if \ \ x \in \mathrm{X_{base}} \\ y, \ if \ (\mathrm{x = y}) \in \mathrm{F_{equ}} \end{cases}$$

Apply the relation to the $\Phi_{\neq}$ of formula (3)

$$(x \neq y) = (\varphi(x) \neq \varphi(y)) \tag{4}$$

as a rewriting rule. As a result, in $\Phi_{\neq}$ we obtain the equalities canonized system of negations, in which $\{x_{j1},...,x_{jl}\} \subset X_0$, $\{z_1,...,z_l\} \subset X_0$. If in this system is an equation in the form $x \neq x$ then the formula (3) is false.

Theorem 3. Formula (3) which is obtained as a result of above transformations, to the formula (1) is the canonical form of (1).

On the analogy of the terminology used above, the canonical form (3) of the formula (1) is designated as a trapezoid off the multiple type. On the analogy of the presentation of the trapezoid off the type *Rat*, formula of trapezoid Φ off the multiple type will be written in the form

$\Phi = < \sigma_1 \cdot ... \cdot \sigma_k \cdot \varepsilon_{k+1} \cdot ... \cdot \varepsilon_n, \delta_1 \& ... \& \delta_l >$,

$\Phi_0 = \sigma_1 \cdot ... \cdot \sigma_k$, $\Phi_= = < \varepsilon_{k+1} \cdot ... \cdot \varepsilon_n$, $\Phi_{\neq} = \delta_1 \& ... \& \delta_l$.

Label the trapezoid $\Phi = < \sigma_1 \cdot ... \cdot \sigma_k \cdot \varepsilon_{k+1} \cdot ... \cdot \varepsilon_{n-1}, \delta_1 \& ... \& \delta_l >$ by $\Phi^{(n-1)}$. Then we can write $\Phi = \Phi^{(n-1)} \cdot \varepsilon_n$.

Operations of union and intersection by trapezoid off the multiple type

Let Φ_1, Φ_2 be the trapezoids off the multiple type. Then the canonical form $\Phi_1 \cap \Phi_2$ is calculated as the algorithm described above, applied to $(\Phi_{0,1} \cdot \Phi_{=,1}) \& (\Phi_{0,2} \cdot \Phi_{=,2})$.

The calculation algorithm of $\Phi_1 \cup \Phi_2$ is the modification of the algorithm of polytrapezoids union. Supposing that $\Phi_1 = \Phi_{0,1} \,\&\, \Phi_{=,1} \,\&\, \Phi_{\neq,1}$, $\Phi_2 = \Phi_{0,2} \,\&\, \Phi_{=,2} \,\&\, \Phi_{\neq,2}$. The projection π_Φ of the trapezoid Φ on the coordinate set Ox_n of variable x_n is equal or A_η, if equality $x_n = A_n$ is an elementary conjunct Φ_0, or A_j, if $(x_j = A_j) \in \Phi_0, (x_n = x_j) \in \Phi_=$. Let us assume that the $\pi_{\Phi_{1,n}}$, $\pi_{\Phi_{2,n}}$ are the projections Φ_1, Φ_2 on Ox_n. Then

$$\Phi_1 \cup \Phi_2 = \Sigma_1 + + \Sigma_2 + + \Sigma_3, \tag{5}$$

where

$$\pi_{\Sigma_{1,n}} = \pi_{\Phi_{1,n}} - \pi_{\Phi_{2,n}}, \quad \pi_{\Sigma_3,n} = \pi_{\Phi_{2,n}} - \pi_{\Phi_{1,n}}, \; \pi_{\Sigma_2,n} = \pi_{\Phi_{1,n}} \cap \pi_{\Phi_{2,n}}, \tag{6}$$

$$\Sigma_{0,1}^{(n-1)} = \Phi_{0,1}^{(n-1)}, \; \Sigma_{=,1} = \Phi_{=.1}, \; \Sigma_{\neq,1} = \Phi_{\neq,1},$$

$$\Sigma_{0,3}^{(n-1)} = \Phi_{0,2}^{(n-1)}, \Sigma_{=,3} = \Phi_{=,2}, \Sigma_{\neq,1} = \Phi_{\neq,3}, \tag{7}$$

$$\Sigma_2^{(n-1} = \Phi_1^{(n-1)} \cup \Phi_2^{(n-1)}. \tag{8}$$

The formulae (5) - (8) represent a recursive algorithm for the computing $\Phi_1 \cup \Phi_2$.

6 Conclusions

The development of systems for the formulae simplifying, which would allow the posed tasks remains an open problem. In addition, no less important is the need to implement appropriate algorithms that makes it possible to simplify formulae at the more efficient level.

The main result of this work is the definition of the canonical form of a linear semi-algebraic formulae that is possessing by the property of uniqueness and by the other useful properties, and the description of its construction algorithm.

In the next article authors plan to describe the implementation of the proposed algorithms by the method of insertion modeling and results of their using in tasks of programs verification.

References

1. CVC4, http://cvc4.cs.nyu.edu/web/
2. Mathsat, http://mathsat.fbk.eu/
3. Cimatti A. et al. The mathSAT5 SMT solver //International Conference on Tools and Algorithms for the Construction and Analysis of Systems. – Springer Berlin Heidelberg, 2013. – C. 93-107.
4. QEPCAD, https://www.usna.edu/CS/qepcadweb/B/WhatisQEPCAD.html
5. Greuel G. M., Pfister G., Schönemann H. Singular 2.0: A Computer Algebra System for Polynomial Computations. University of Kaiserslautern (2001).
6. Singular, https://www.singular.uni-kl.de/

7. COCOA, http://cocoa.dima.unige.it/
8. COCOA, http://cocoa.dima.unige.it/cocoalib/
9. MINIZINC, http://www.minizinc.org/
10. STP constraint solver, http://stp.github.io/
11. Redlog, http://www.redlog.eu/
12. Satallax, http://www.ps.uni-saarland.de/~cebrown/satallax/
13. Isabelle, https://isabelle.in.tum.de/
14. E-SETHEO, http://www4.informatik.tu-muenchen.de/~schulz/WORK/e-setheo.html
15. MiniSat, http://minisat.se/
16. SMTInterpol, http://ultimate.informatik.uni-freiburg.de/smtinterpol/
17. About SMTInterpol, http://gtps.math.cmu.edu/tps-about.html
18. Theorem-proving system, http://dic.academic.ru/dic.nsf/eng_rus/793528/theorem
19. Gandalf, http://deepthought.ttu.ee/it/gandalf/
20. Vampire, http://www.vprover.org/download.cgi
21. Vampire 4.1-SMT System Description, http://smtcomp.sourceforge. net/ 2016/systemDescriptions/ Vampire.pdf
22. QEPCAD - The Solution Formula Construction Phase, https://www.usna.edu/CS/qepcadweb/B/user/Solution.html
23. The E Theorem Prover, http://wwwlehre.dhbw-stuttgart.de/~sschulz/E/E.html
24. iProver, http://www.cs.man.ac.uk/~korovink/iprover/
25. LEO-II, http://page.mi.fu-berlin.de/cbenzmueller/leo/
26. Z3Prover, https://github.com/Z3Prover/z3
27. ABOUT WALDMEISTER, http://www.waldmeister.org/
28. SPASS, http://www.mpi-inf.mpg.de/departments/automation-of-logic/software/spass-workbench/
29. M. S. Lvov, “Algebraic approach to the problem of solving systems of linear inequalities,” Cybernetics and Systems Analysis, 46, No. 2, 326–338 (2010).
30. Lvov, M. S., and V. S. Peschanenko. "Trapezoid method for solving systems of linear inequalities and its implementation in insertion modeling." Cybernetics and Systems Analysis 48, No. 6, 931-942. (2012)
31. J. Goguen and J. Meseguer, “Ordered-sorted algebra I: Partial and Overloaded Operations, Errors and Inheritance,” Computer Science, 105, No. 2, 217–273 (1992).
32. J. A. Goguen, J. W. Thatcher, and E. Wagner, “An initial algebra approach to the specification, correctness and implementation of abstract data types,” in: R. Yeh (ed.), Current Trends in Programming Methodology, Prentice Hall, NJ (1978), pp. 80–149.
33. M. S. Lvov, “Synthesis of interpreters of algebraic operations in extensions of multisorted algebras,” Visn. Khark. Nats. Un-tu, No. 847, 221–238 (2009).
34. M. S. Lvov, “Verification of interpreters of algebraic operations in extensions of multisorted algebras,” KhUPS, No. 3(21), 127–137 (2009).
35. M. S. Lvov, “Method of inheritance in implementing algebraic computations in mathematical systems of educational destination,” Syst. Upravl., Navigats. 3 Zv’yazku, TsNDINiU, No. 3 (11), 120–130, (2009).
36. M. S. Lvov, “Method of morphisms in implementing algebraic computations in mathematical systems of educational destination,” Systemy Obrobky Informatsii, No. 6 (80), 183–190 (2009)
37. M. S. Lvov, “An approach to the implementation of algebraic computations: Computations in propositional algebra,” Visn. Khark. Nats. Un-tu, Information Technologies: Mathematical Modeling, No. 863, 157–168 (2009).

38. T. S. Motzkin, H. Raiffa, G. L. Thompson, and R. M. Thrall “The double description method” in: Matrix Games, Fizmatgiz, Moscow (1961), pp. 81–109.
39. G. L. Zeidler, Lectures on Convex Polytopes, Springer, Berlin–New York (1994).
40. S. N. Tchernikov, Linear Inequalities [in Russian], Nauka, Moscow (1968).
41. Z3, http://rise4fun.com/z3/tutorial/guide
42. De Moura, Leonardo, and Nikolaj Bjørner. "Z3: An efficient SMT solver." International conference on Tools and Algorithms for the Construction and Analysis of Systems. Springer Berlin Heidelberg (2008).