Computing and Informatics, Vol. 31, 2012, 1295–1328

ALGEBRAIC APPROACH TO LOGICAL INFERENCE IMPLEMENTATION

Boris Kulik

Institute of Problems in Machine Science
Russian Academy of Sciences (RAS)
61 Bolshoi pr., 199178 St. Petersburg, Russia
e-mail: `ba-kulik@yandex.ru`

Alexander Fridman, Alexander Zuenko

Institute for Informatics and Mathematical Modelling
Kola Science Centre of RAS
24A Fersman str., 184209 Apatity, Russia
e-mail: {`fridman, zuenko`}`@iimm.kolasc.net.ru`

Communicated by Ivan Plander

Abstract. The paper examines the usage potential of n-tuple algebra (NTA) developed by the authors as a theoretical generalization of structures and methods applied in intelligence systems. NTA supports formalization of a wide set of logical problems (abductive and modified conclusions, modelling of graphs, semantic networks, expert rules, etc.). This article mostly describes implementation of logical inference by means of NTA. Logical inference procedures in NTA can include, besides the known logical calculus methods, new algebraic methods for checking correctness of a consequence or for finding corollaries to a given axiom system. Inference methods consider (above feasibility of certain substitutions) inner structure of knowledge to be processed, thus providing faster solving of standard logical analysis tasks. Matrix properties of NTA objects allow to decrease laboriousness of intellectual procedures as well as to efficiently parallel logical inference algorithms. In NTA, we discovered new structural and statistical classes of conjunctive normal forms whose satisfiability can be detected for polynomial time. Consequently, many algorithms whose complexity evaluation is theoretically high, e.g. exponential, can in practice be solved in polynomial time, on the average. As for making databases more intelligent, NTA can be considered an extension of relational algebra to knowledge processing. In the authors' opinion, NTA can become a methodological basis for creating knowledge processing languages.

Keywords: Data processing, knowledge representation, intelligence system, multi-place relation, general theory of relations, n-tuple algebra, flexible universe, logical inference, knowledge processing language, parallel computing

1 INTRODUCTION

Developers of modern intelligence systems face certain challenges resulting from fundamentally different approaches used in constructing databases (DB) and knowledge bases (KB). KB design is based on a mathematical system that is named by a number of terms: formal approach, axiomatic method, symbolic logic, theory of formal systems (TFS). Development of TFS began in the works of B. Russell, L. Wittgenstein, D. Hilbert, G. Peano and others at the beginning of 20th century when paradoxes of set theory were discovered and the algebra of sets and Boolean algebra were no longer the most important approaches to foundations of logic.

In TFS, inference rules are defined in the way that allows to interpret new symbol constructions as corollaries to or new theorems from the symbol constructions or statements that are axioms or theorems in the given formal system.

Additionally, in TFS we need to reduce many logical analysis tasks to satisfiability checks for a certain logical formula, this check being able to return only two possible answers ("yes" or "no"). Despite a substantial number of positive results that have been obtained in this field, such a reduction is not sufficiently simple yet. Moreover, the reduction is unrealizable in cases when we need not only to receive a "yes/no" answer but also to estimate the value of some parameters in the formal system or to assess the structure and/or number of objects that satisfy the given conditions. That is why artificial intelligence languages based on declarative approach grew much more complicated due to the necessity of furnishing them with different non-declarative procedures and functions.

Today, mathematical logic is based on strict rules of pure calculus. This calculus has been proven to be isomorphic to some algebraic systems; for instance, propositional calculus is isomorphic to Boolean algebra. However, algebraic (procedural) approach is fairly seldom used by itself in theoretical research on classical logic today. On the other hand, algebraic methods are widely used in applied research, particularly in software implementation of mentioned non-declarative functions in intelligence systems.

Algebraic techniques, e.g. those of relational algebra are most commonly used in constructing data processing systems. Note that the term "data processing languages" (DPL) is very popular in data management while intelligence systems mostly deal with knowledge representation languages (KRL). This shows the declarative origin of KRLs and the procedural basis of DPLs. In other words, DPLs regulate the way actions are performed on data, whereas KRLs specify what is to be done with the knowledge without determining how to do this. Thus, algebraic approach seems to be a rational supplement to traditional formal methods in logic for improving logical analysis techniques and creating knowledge processing languages (KPL) that allow to flexibly program and compare algorithms for intelligent procedures.

Methodical differences in constructing DBs and KBs make using them within a single integrated software system complicated. This problem was first posed at the IJCAI '95 (The International Joint Conference on Artificial Intelligence in Montreal, Canada on August 19 through 25, 1995) and now it becomes even more topical as making database management systems (DBMS) more intelligent by developing DB semantic interfaces, deductive DBs, etc., becomes more important. This is why developing a unified methodology of data and knowledge processing is required. In our opinion, this can be achieved through algebraic methods if the concept of multiplace relation is used as a base concept. This idea allows to represent many data and knowledge systems not only as an artificial language, but also as a totality of relations with different diagrams that are subject to certain operations similar to those of algebra of sets.

Below we briefly describe conventional mathematical sections dealing with relations, and propose a mathematical system named n-tuple algebra (NTA) [9, 11] and developed for solving the set of problems described above [13, 21]. We believe that NTA can be used as a base for creating knowledge processing languages.

2 RELATIONS IN MATHEMATICS

A general theory of relations has not been developed yet. The term "theory of relations" is usually used either for theory of binary relations or for theory of multiplace relations based on relational algebra. In any case, these theories accept the classical mathematical definition of a relation through Cartesian product. If D is a Cartesian product of n different or equal sets, then an *n-place relation* R is a certain subset of elementary n-tuples contained in D.

Such a definition of a multiplace relation allows treating relations as ordinary sets if they are defined on the same Cartesian product D. If so, the *complement of a multiplace relation* R is the set difference $D \setminus R$. For example, if

$$D = \{1, 2, 3, 4, 5\} \times \{1, 2, 3, 4, 5\},$$

and R is a relation "less than" in the set of numbers $\{1, 2, 3, 4, 5\}$, then after eliminating all elementary n-tuples belonging to R, from D we get a set of elementary n-tuples corresponding to the relation "more than or equal to".

However, this conformity between algebra of multiplace relations and algebra of sets is no longer valid in totalities of relations defined on various Cartesian products since it is impossible to determine operations of union and intersection for these. Besides, algebra of multiplace relations includes operations of composition and join that have no equivalent operations in algebra of sets.

According to the classical definition, a multiplace relation is a set of elementary n-tuples; however, this definition is not always practical since it results in redundancy due to multiple copying the same data to memory. Let us consider as an example a relation which reflects the fact that a professor *Smith* teaches subjects *Mathematics*, *Logic*, and *Physics*: $\{(\text{Smith}, \text{Mathematics}), (\text{Smith}, \text{Logic}), (\text{Smith}, \text{Physics})\}$. This relation can be compacted as a Cartesian product $\{\text{Smith}\} \times \{\text{Mathematics}, \text{Logic}, \text{Physics}\}$. Obviously, not every relation can be represented as a single Cartesian product composed of non-elementary sets. For example, the following relation cannot be expressed this way:

$$P = \begin{bmatrix} \text{Smith} & \text{Mathematics} \\ \text{Smith} & \text{Logic} \\ \text{Smith} & \text{Physics} \\ \text{Burns} & \text{Logic} \\ \text{Burns} & \text{Philosophy} \end{bmatrix}$$

Nevertheless every relation can be represented as a union of certain Cartesian products that are, in a general case, composed of subsets of corresponding attributes. In our example, this union is $P = \{\text{Smith}\} \times \{\text{Mathematics}, \text{Logic}, \text{Physics}\} \cup \{\text{Burns}\} \times \{\text{Logic}, \text{Philosophy}\}$. Transition from elementary n-tuples to ones composed of sets rather than elements provides a significant reduction in computational resources used for processing relations (calculating unions, intersections, complements, etc.) and data storing.

Theoretical basics of mathematical logic are set forth in formal language of predicate calculus [16]; but interpretation of logic uses a system that can be represented as a totality of relations with elements being sequences of symbols. The formulas of mathematical logic can be expressed as sets of satisfiable substitutions, i.e. relations as well.

There exist different descriptive languages in artificial intelligence; but, as a rule, we can transform examples introduced in publications for illustrating different methods and approaches to structures like $N(E_1, E_2, \ldots)$ where N is a name of a relation or a predicate and E_1, E_2, $\ldots$ are names of objects belonging to certain totalities of values of properties (attributes) [17]. Operations on such structures completely correspond to those of theory of relations.

Apart from this, an initial relation defined on a Cartesian product $D = X_1 \times X_2 \times \ldots \times X_n$ can be often split into blocks corresponding to relations on some projections of D, which greatly reduces laboriousness of operations on this relation by using its matrix properties. This allows to process every block separately using known features of Cartesian products, for instance, by paralleling the necessary operations.

Since blocks reflect relations defined on different diagrams, it is necessary to provide specific algebraic operations to recover the initial relation from the blocks.

This article introduces n-tuple algebra that uses Cartesian product of sets rather than sequences of elements (elementary n-tuples) as a basic structure, and implements the general theory of multiplace relations. NTA supports formalization of a wide set of logical problems (abductive and modified conclusions, modelling of graphs, semantic networks, expert rules, etc. [13, 19]). Below we will focus on performing logical inference by means of NTA.

"Compacted" representation of relations allows to apply algebraic approach not only in database management systems, but in knowledge systems as well, as it reduces computational laboriousness of logical inference in many cases. As for making DBs more intelligent, NTA can be considered an extension of relational algebra to knowledge processing.

3 BASICS OF N-TUPLE ALGEBRA

3.1 Basic Concepts and Structures

N-tuple algebra was developed for modeling and analysis of multiplace relations. Unlike relational algebra used for formalization of databases NTA can use all mathematical logic's means for logic modeling and analysis of systems, namely logical inference, corollary trueness' check, analysis of hypotheses, abductive inference, etc. N-tuple algebra is based on the known properties of Cartesian products of sets which correspond to the fundamental laws of mathematical logic. In NTA, transitional results can be obtained without representation of structures as sets of elementary n-tuples since every NTA operation uses sets of components of attributes or n-tuples of components.

Definition 1. *N-tuple algebra* is an algebraic system whose support is an arbitrary set of multiplace relations expressed by specific structures, namely elementary n-tuple, C-n-tuple, C-system, D-n-tuple, and D-system, called n-tuple algebra objects. So, apart from the elementary n-tuple, NTA contains four more structures providing a compact expression for sets of elementary n-tuples.

Names of NTA objects consist of a name proper, sometimes appended with a string of names of attributes in square brackets; these attributes determine the relation diagram in which the n-tuple is defined. For instance, if an elementary n-tuple $T[XYZ] = (a, b, c)$ is given, then T is the name of the elementary n-tuple (a, b, c), X, Y, Z are names of attributes, and $[XYZ]$ is the relation diagram (i.e. space of attributes), $a \in X$, $b \in Y$ and $c \in Z$. A domain is a set of all values of an attribute. Domains of attributes correspond to definitional domains of variables in mathematical logic, and to scales of properties in information systems. Hereafter attributes are denoted by capital Latin letters which may sometimes have indices, and the values of these attributes are denoted by the same lower-case Latin letters. A set of attributes representing the same domain is called a sort. Structures defined on the same relation diagram are called homotypic ones. Any totality of homotypic NTA objects is an algebra of sets.

N-tuple algebra is based on the concept of a flexible universe. A flexible universe consists of a certain totality of partial universes that are Cartesian products of domains for a given sequence of attributes. A relation diagram determines a certain partial universe.

In a space of properties $\boldsymbol{S}$ with attributes X_i (i.e. $\boldsymbol{S} = X_1 \times X_2 \times \ldots \times X_n$), the flexible universe will be comprised of different projections i.e. subspaces that use a part of attributes from $\boldsymbol{S}$. Every such subspace corresponds to a partial universe.

Definition 2. An *elementary n-tuple* is a sequence of elements each belonging to the domain of the corresponding attribute in the relation diagram. An example of an elementary n-tuple $T[XYZ]$ is given above.

Definition 3. A *C-n-tuple* is an n-tuple of sets (components) defined in a certain relation diagram; each of these sets is a subset of the domain of the corresponding attribute.

A C-n-tuple is a set of elementary n-tuples; this set can be enumerated by calculating the Cartesian product of the C-n-tuple's components. C-n-tuples are denoted with square brackets. For example, $R[XYZ] = [ABC]$ means that $A \subseteq X$, $B \subseteq Y$, $C \subseteq Z$ and $R[XYZ] = A \times B \times C$.

Definition 4. A *C-system* is a set of homotypic C-n-tuples that are denoted as a matrix in square brackets. The C-n-tuples that such a matrix contains are rows of this matrix.

A C-system is a set of elementary n-tuples. This set equals to the union of sets of elementary n-tuples that the corresponding C-n-tuples contain. For example, a C-system $Q[XYZ] = \begin{bmatrix} A_1 & B_1 & C_1 \\ A_2 & B_2 & C_2 \end{bmatrix}$ can be represented as a set of elementary n-tuples calculated by formula $Q[XYZ] = (A_1 \times B_1 \times C_1) \cup (A_2 \times B_2 \times C_2)$.

In order to combine relations defined on different projections within a single algebraic system isomorphic to algebra of sets, NTA introduces dummy attributes formed by using dummy components. There are two types of these components. One of them called a complete component is used in C-n-tuples and is denoted by "*". A dummy component "*" added in the i^{th} place in a C-n-tuple or in a C-system equals to the set corresponding to the whole range of values of the attribute X_i. In other words, the domain of this attribute is the value of the dummy component. For example, if the domain of attribute X is given (here it equals to the set $\{a, b, c, d\}$), the C-n-tuple $Q[YZ] = [\{f, g\}\,\{a, c\}]$ can be expressed in the relation diagram $[XYZ]$ as a C-n-tuple $[*\{f, g\}\,\{a, c\}]$. Since the dummy component of Q corresponds to an attribute with the domain X, the equality $[*\{f, g\}\,\{a, c\}] = [\{a, b, c, d\}\,\{f, g\}\,\{a, c\}]$ is true. Another dummy component ($\emptyset$) called an empty set is used in D-n-tuples.

A C-n-tuple that has at least one empty component is empty. In NTA, if we deal with models of propositional or predicate calculuses, this statement is accepted as an axiom which has an interpretation based on the properties of Cartesian products.

Below, we will show that usage of dummy components and attributes in NTA allows to transform relations with different relation diagrams into ones of the same type, and then to apply operations of theory of sets to these transformed relations.

The proposed technique of defining dummy attributes differs from the known techniques essentially due to the fact that new data are inputted into multiplace relations as sets rather than elementwise, which significantly reduces both computational laboriousness and memory capacity for representation of the structures.

Operations (intersection, union, complement) and checks of relations of inclusion or equality for these NTA objects are based on Theorems 1–6. Here they are given without proof because their formulating in terms of NTA corresponds to the known properties of Cartesian products. Let two homotypic C-n-tuples $P = [P_1P_2 \dots P_n]$ and $Q = [Q_1Q_2 \dots Q_n]$ be given.

Theorem 1. $P \cap Q = [P_1 \cap Q_1\, P_2 \cap Q_2 \dots P_n \cap Q_n]$.

Example 1. $[\{b, d\}\, \{f, h\}\, \{a, b\}] \cap [*\{f, g\}\{a, c\}] = [\{b, d\}\, \{f\}\, \{a\}]$;
$[\{b, d\}\, \{f, h\}\{a, b\}] \cap [*\{g\}\, \{a, c\}] = [\{b, d\}\, \emptyset\, \{a\}] = \emptyset$.

Theorem 2. $P \subseteq Q$, if and only if $P_i \subseteq Q_i$ for all $i = 1, 2, \dots, n$.

Theorem 3. $P \cup Q \subseteq [P_1 \cup Q_1\, P_2 \cup Q_2 \dots P_n \cup Q_n]$, equality being possible only in two cases:

1. $P \subseteq Q$ or $Q \subseteq P$;
2. $P_i = Q_i$ for all corresponding pairs of components except one pair.

Note that in NTA, according to Definition 4, equality $P \cup Q = \begin{bmatrix} P_1 & P_2 & \dots & P_n \\ Q_1 & Q_2 & \dots & Q_n \end{bmatrix}$ is true for all cases.

Theorem 4. Intersection of two homotypic C-systems equals to a C-system that contains all non-empty intersections of each C-n-tuple of the first C-system with each C-n-tuple of the second C-system.

Example 2. Let the following two C-systems be given in space S:

$$R_1[XYZ] = \begin{bmatrix} \{a, b, d\} & \{f, h\} & \{b\} \\ \{b, c\} & * & \{a, c\} \end{bmatrix},$$

$$R_2[XYZ] = \begin{bmatrix} \{a, d\} & * & \{b, c\} \\ \{b, d\} & \{f, h\} & \{a, c\} \\ \{b, c\} & \{g\} & \{b\} \end{bmatrix}.$$

We need to calculate their intersection. First we calculate intersection of all the pairs of C-n-tuples that the two different C-systems contain:

$$\begin{aligned} [\{a, b, d\}\, \{f, h\}\, \{b\}] \cap [\{a, d\} * \{b, c\}] &= [\{a, d\}\, \{f, h\}\, \{b\}]; \\ [\{a, b, d\}\, \{f, h\}\, \{b\}] \cap [\{b, d\}\, \{f, h\}\, \{a, c\}] &= \emptyset; \\ [\{a, b, d\}\, \{f, h\}\, \{b\}] \cap [\{b, c\}\, \{g\}\, \{b\}] &= \emptyset; \\ [\{b, c\} * \{a, c\}] \cap [\{a, d\} * \{b, c\}] &= \emptyset; \end{aligned}$$

$$\begin{aligned}[\{b,c\} * \{a,c\}] \cap [\{b,d\}\,\{f,h\}\,\{a,c\}] &= [\{b\}\,\{f,h\}\,\{a,c\}];\\ [\{b,c\} * \{a,c\}] \cap [\{b,c\}\,\{g\}\,\{b\}] &= \emptyset.\end{aligned}$$

Then we form a C-system from non-empty C-n-tuples:

$$R_1 \cap R_2 = \begin{bmatrix} \{a,d\} & \{f,h\} & \{b\} \\ \{b\} & \{f,h\} & \{a,c\} \end{bmatrix}.$$

Theorem 5. Union of two homotypic C-systems equals to a C-system that contains all C-n-tuples of the operands.

After calculating the union of the C-systems, the total number of n-tuples in the derived C-system can be reduced in some cases by using conditions 1. or 2. of Theorem 3.

In order to introduce the algorithms for calculating complements of NTA objects, we need one more definition.

Definition 5. A complement $(\overline{P_j})$ of any component P_j of an NTA object is defined as a complement to the domain of the attribute corresponding to this component.

For example, if a C-n-tuple $R[XYZ] = [ABC]$ is given, then $\overline{A} = X \setminus A$, $\overline{B} = Y \setminus B$ and $\overline{C} = Z \setminus C$.

Theorem 6. For an arbitrary C-n-tuple $P = [P_1 P_2 \dots P_n]$

$$\overline{P} = \begin{bmatrix} \overline{P_1} & * & \dots & * \\ * & \overline{P_2} & \dots & * \\ \dots & \dots & \dots & \dots \\ * & * & \dots & \overline{P_N} \end{bmatrix}$$

In the above C-system $\overline{P}$ whose dimension is $n \times n$, all the components except the diagonal ones are dummy components. We shall call such C-systems *diagonal C-systems.*

Here is an example. Let a C-n-tuple $T = [\{b,d\}\,\{f,h\}\,\{a,b\}]$ be given in the space $\boldsymbol{S} = X \times Y \times Z$ where $X = \{a,b,c,d\}$, $Y = \{f,g,h\}$, $Z = \{a,b,c\}$. Then

$$\overline{T} = \begin{bmatrix} X \setminus \{b,d\} & * & * \\ * & Y \setminus \{f,h\} & * \\ * & * & Z \setminus \{a,b\} \end{bmatrix} = \begin{bmatrix} \{a,c\} & * & * \\ * & \{g\} & * \\ * & * & \{c\} \end{bmatrix}$$

We can denote diagonal C-systems as one n-tuple of sets, using reversed square brackets for expressing this. Then we get the following equality: $\overline{T} =]\{a,c\}\,\{g\}\,\{c\}[$.

Such a "reduced" expression for a diagonal C-system makes up a new NTA structure called a D-n-tuple.

Definition 6. A *D-n-tuple* is an n-tuple of components enclosed in reversed square brackets which equals a diagonal C-system whose diagonal components equal the corresponding components of the D-n-tuple.

The complement of a C-n-tuple can be directly recorded as a D-n-tuple. For example, if $T_1 = [\{b, d\} * \{a, b\}]$, then $\overline{T_1} =]\{a, c\}\ \emptyset\ \{c\}[$. In D-n-tuples the constant "$\emptyset$" is a dummy component.

This structure not only allows to compactly denote diagonal C-systems, but can be also used in some operations and retrieval queries. The terms C-n-tuple and D-n-tuple were chosen due to the following reason: if we represent the components of these n-tuples as predicates, C-n-tuple corresponds to conjunction of these predicates, and D-n-tuple corresponds to disjunction of these predicates. D-n-tuples are used to form one more NTA structure, namely a D-system.

Definition 7. A *D-system* is a structure that consists of a set of homotypic D-n-tuples and equals the intersection of sets of elementary n-tuples that these D-n-tuples contain.

Expression for a D-system is analogous to that of a C-system except that in this case reversed square brackets are used instead of the regular ones.

Theorem 7. The complement of a C-system is a D-system of the same dimension, in which each component is equal to the complement of the corresponding component in the initial C-system.

Proof. Let a C-system P that contains a set $\{P_1, P_2, \ldots, P_n\}$ of C-n-tuples be given. This means that $P = P_1 \cup P_2 \cup \ldots \cup P_n$. Calculating its complement according to de Morgan's law, we get the following result: $\overline{P} = \overline{P_1} \cap \overline{P_2} \cap \ldots \cap \overline{P_n}$. Then the validity of this theorem follows from the Theorem 6 and Definitions 6 and 7. □

For example, the complement of a C-system

$$F[XYZ] = \begin{bmatrix} \{a, b, d\} & \{f, h\} & \{b\} \\ \{b, c\} & * & \{a, c\} \end{bmatrix}$$

given in a space $\boldsymbol{S}$ can be calculated as a D-system

$$\overline{F} = \left] \begin{matrix} X \setminus \{a, b, d\} & Y \setminus \{f, h\} & Z \setminus \{b\} \\ X \setminus \{b, c\} & Y \setminus * & Z \setminus \{a, c\} \end{matrix} \right[= \left] \begin{matrix} \{c\} & \{g\} & \{a, c\} \\ \{a, d\} & \emptyset & \{b\} \end{matrix} \right[.$$

It is easy to see that relations between C-objects (C-n-tuples and C-systems) and D-objects (D-n-tuples and D-systems) are in accordance with de Morgan's laws of duality. Due to this fact, they are called *alternative classes*. Calculation of the complement for an NTA object always has polynomial computational complexity. Operations of union and intersection have polynomial complexity for NTA objects belonging to the same class, but a transformation into an alternative class is also necessary for objects of different classes.

For implementing intelligence systems, it is often necessary to transform NTA objects into an alternative class. Complexity of this transformation will be discussed below in Section 5. Let us now introduce theorems regulating this transformation.

Theorem 8. Every C-n-tuple (D-n-tuple) P can be transformed into an equivalent D-system (C-system) in which every non-dummy component p_i corresponding to an attribute X_i of the initial n-tuple is expressed by a D-n-tuple (C-n-tuple) having the component p_i in the attribute X_i and dummy components in all the rest attributes.

Proof. The statement regarding transformation a D-n-tuple into a C-system immediately follows from the definition of a D-n-tuple as a compact expression for the corresponding C-system. The algorithm of transformation of a C-n-tuple into an equivalent D-system results from the duality property of alternative classes. $\square$

For example, a D-n-tuple $]A\,\emptyset\,BC[$ where A, B, C are not dummy can be recorded as a C-system $\begin{bmatrix} A & * & * & * \\ * & * & B & * \\ * & * & * & C \end{bmatrix}$, and a C-n-tuple $[AB * C]$ – as a D-system $\left]\begin{matrix} A & \emptyset & \emptyset & \emptyset \\ \emptyset & B & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & C \end{matrix}\right[$.

Evidently, algorithms for transformation of C-n-tuples and D-n-tuples into structures of an alternative class are not exponentially complex. Laboriousness of the algorithms increases significantly for C-systems and D-systems. Two following assertions are given here without any proof due to their obviousness.

Theorem 9. A D-system P containing m D-n-tuples is equivalent to a C-system equal to the intersection of m C-systems obtained by transformation every D-n-tuple belonging to P into a C-system.

Theorem 10. A C-system P containing m C-n-tuples is equivalent to a D-system equal to the union of m D-systems obtained by transforming every C-n-tuple belonging to P into a D-system.

Transformations of NTA objects into objects of alternative classes allow to realize all operations of theory of sets on NTA objects, as well as all checks of relations among such objects without having to represent the objects as sets of elementary n-tuples. In some cases, inclusion checks can be done directly for structures belonging to different alternative classes. The following theorems describe these cases.

Theorem 11. $P \subseteq Q$ is true for a C-n-tuple $P = [p_1 p_2 \dots p_n]$ and a D-n-tuple $Q =]q_1 q_2 \dots q_n[$ if and only if $p_i \subseteq q_i$ is true for at least one value of i.

Proof. A D-n-tuple is equivalent to a C-system containing n C-n-tuples all of whose components are complete dummy components except q_i. So, the necessity of the theorem statement follows from the fact that a C-system is a union of the

C-n-tuples. Indeed, if one of the C-n-tuples Q_i belonging to the C-system obtained after transforming the initial D-n-tuple Q equals to $[**\ldots q_i \ldots *]$ and $p_i \subseteq q_i$, then $P \subseteq Q_i$ and hence $P \subseteq Q$. Let us prove the sufficiency. Suppose $p_i \subseteq q_i$ is false for every i. We need to prove that $P \subseteq Q$ is impossible then. This supposition lets us conclude that for every i, there is a $r_i = p_i \setminus q_i \neq \emptyset$. Consequently, $r_i \subseteq p_i$ and $r_i \subseteq \overline{q_i}$ for every i. Then, a non-empty C-n-tuple $R = [r_1 r_2 \ldots r_n]$ exists for which $R \subseteq P$ and $R \subseteq \overline{Q}$ that proves impossibility of $P \subseteq Q$ is. □

Theorem 12. $P \subseteq Q$ is true for a C-n-tuple P and a D-system Q if and only if $P \subseteq Q_j$ is true for every D-n-tuple Q_j belonging to Q.

Proof. A D-system is an intersection of sets comprising all elementary n-tuples from D-n-tuples contained in the D-system, then, if P is included in every D-n-tuple, it is included in their intersection i.e. in the D-system. □

We have already mentioned that NTA allows performing operations of algebra of sets on homotypic (having the same relation diagram) NTA objects only. In order to perform these on multiplace relations defined on different diagrams, we need to transform them into ones of the same diagram. For this, NTA has 5 more operations on attributes, namely:

1. renaming of attributes;
2. transposition of attributes and corresponding columns in NTA objects;
3. inversion of NTA objects (for binary relations);
4. addition of a dummy attribute (+Attr);
5. elimination of an attribute (-Attr).

Below we introduce these operations and some derivative ones used in logical inference.

3.2 Operations with Attributes, Join and Composition Operations, Generalized Operations

Renaming of attributes is only possible for attributes of the same sort. This operation is used when it is necessary to substitute variables, particularly, in algorithms for calculating transitive closure of a graph.

Transposition of attributes is an operation that swaps columns in an NTA object's matrix and respectively changes the order of attributes in the relation diagram.

This operation does not change the content of the relation. The operation is used for transforming NTA objects whose attributes are the same, but come in different order to a form that allows performing algebra of sets' operations on them.

For example, a C-system $P[XYZ] = \begin{bmatrix} \{a,b,d\} & \{f,h\} & \{b\} \\ \{b,c\} & * & \{a,c\} \end{bmatrix}$ transforms into a C-system $P[YXZ] = \begin{bmatrix} \{f,h\} & \{a,b,d\} & \{b\} \\ * & \{b,c\} & \{a,c\} \end{bmatrix}$ due to transposition of attributes.

Inversion of NTA objects. In case of binary relations, swapping columns without swapping attributes allows to get the relation inverse to the initial one. For example, swapping columns of relation $G[XY] = \begin{bmatrix} \{a\} & \{a,b\} \\ \{b,c\} & \{a,c\} \end{bmatrix}$ turns it into the inverse relation $G^{-1}[XY] = \begin{bmatrix} \{a,b\} & \{a\} \\ \{a,c\} & \{b,c\} \end{bmatrix}$. In this case, inversion of an NTA object turns all the elementary n-tuples (s,t) of the initial relation into the inverse ones (t,s). If an elementary n-tuple contains identical elements (e.g. (b,b)), it does not change during the inversion.

Addition of a dummy attribute (+Attr) is done when the added attribute is missing in the relation diagram of an NTA object (NTA objects with duplicate attributes are also possible, but are not considered here). This operation simultaneously adds the name of a new attribute into the relation diagram and adds a new column with dummy components into the corresponding place; dummy components "*" are added into C-n-tuples and C-systems, and dummy components "∅" are added into D-n-tuples and D-systems.

Elimination of an attribute (-Attr) is done in the following way: a column is removed from an NTA object, and the corresponding attribute is removed from the relation diagram.

Semantics of the +Attr and -Attr operations will be explained below in Section 4. These operations are used, in particular, for calculating join or composition of two different-type relations defined by NTA objects. In general case, join and composition operations of relations can be performed for any pairs of NTA objects. Let two structures $R_1[\boldsymbol{V}]$ and $R_2[\boldsymbol{W}]$ be given, where $\boldsymbol{V}$ and $\boldsymbol{W}$ are sets of attributes and $\boldsymbol{V} \neq \boldsymbol{W}$. These sets can be separated into nonintersecting subsets with the following transformations:

$$\boldsymbol{X} = \boldsymbol{W} \setminus \boldsymbol{V}; \quad \boldsymbol{Y} = \boldsymbol{W} \cap \boldsymbol{V}; \quad \boldsymbol{Z} = \boldsymbol{V} \setminus \boldsymbol{W}.$$

Then we get $\boldsymbol{V} = \boldsymbol{Y} \cup \boldsymbol{Z}$ and $\boldsymbol{W} = \boldsymbol{X} \cup \boldsymbol{Y}$. Taking this into account, the given relations can be expressed as follows: $R_1[\boldsymbol{Y}\,\boldsymbol{Z}]$ and $R_2[\boldsymbol{X}\,\boldsymbol{Y}]$.

Join operation $R_1[\boldsymbol{Y}\,\boldsymbol{Z}] \oplus R_2[\boldsymbol{X}\,\boldsymbol{Y}]$ for relations is usually done by pairwise comparison of all elementary n-tuples from different relations. If comparing these n-tuples shows that they coincide in the projection $[\boldsymbol{Y}]$, an n-tuple with relation diagram $[\boldsymbol{X}\,\boldsymbol{Y}\,\boldsymbol{Z}]$ is formed from the two n-tuples, the new n-tuple becoming one of the elements of the relational join. For example, there are two elementary n-tuples

$T_1 \in R_1$ and $T_2 \in R_2$, where

$$T_1[\boldsymbol{Y}\,\boldsymbol{Z}] = (c, d, e, f, g); \quad T_2[\boldsymbol{X}\,\boldsymbol{Y}] = (a, b, c, d, e),$$

and

$$T_2[\boldsymbol{X}] = (a, b); \quad T_1[\boldsymbol{Z}] = (f, g); \quad T_1[\boldsymbol{Y}] = T_2[\boldsymbol{Y}] = (c, d, e).$$

Then the result of join of these n-tuples is the elementary n-tuple $T_3[\boldsymbol{X}\,\boldsymbol{Y}\,\boldsymbol{Z}] = (a, b, c, d, e, f, g)$.

In NTA relational join operation is substantially simplified and can be calculated without pairwise comparison of all elementary n-tuples using the following formula:

$$R_1[\boldsymbol{Y}\,\boldsymbol{Z}] \oplus R_2[\boldsymbol{X}\,\boldsymbol{Y}] = +\boldsymbol{X}(R_1) \cap +\boldsymbol{Z}(R_2). \tag{1}$$

Operation of composition $R_1[\boldsymbol{Y}\,\boldsymbol{Z}] \circ R_2[\boldsymbol{X}\,\boldsymbol{Y}]$ of relations is performed after calculating their join. For this, we need to eliminate the projection $[\boldsymbol{Y}]$ from all elementary n-tuples belonging to the join. For example, an elementary n-tuple $T_4[\boldsymbol{X}\,\boldsymbol{Z}] = (a, b, f, g)$ is the composition of the two n-tuples T_1 and T_2 considered above.

In NTA, the composition of relations is calculated according to the formula:

$$R_1[\boldsymbol{Y}\,\boldsymbol{Z}] \circ R_2[\boldsymbol{X}\,\boldsymbol{Y}] = -\boldsymbol{Y}(+\boldsymbol{X}(R_1) \cap +\boldsymbol{Z}(R_2)) = -\boldsymbol{Y}(R_1 \oplus R_2), \tag{2}$$

if $(R_1 \oplus R_2)$ is a C-n-tuple or a C-system.

Here is an example. Let the following NTA objects be given in space $\boldsymbol{S}$:

$$R_1[YZ] = \begin{bmatrix} \{f\} & \{a, b\} \\ \{g, h\} & \{a, c\} \end{bmatrix}; \quad R_2[XY] = \begin{bmatrix} \{a\} & \{g, h\} \\ \{b, c\} & \{f\} \end{bmatrix}$$

Let us calculate join of these relations by formula (1):

$$R_1 \oplus R_2 = \begin{bmatrix} * & \{f\} & \{a, b\} \\ * & \{g, h\} & \{a, c\} \end{bmatrix} \cap \begin{bmatrix} \{a\} & \{g, h\} & * \\ \{b, c\} & \{f\} & * \end{bmatrix} = \begin{bmatrix} \{b, c\} & \{f\} & \{a, b\} \\ \{a\} & \{g, h\} & \{a, c\} \end{bmatrix}$$

Then we calculate their composition in the relation diagram $[XZ]$ by formula (2):

$$R_1 \circ R_2 = \begin{bmatrix} \{b, c\} & \{a, b\} \\ \{a\} & \{a, c\} \end{bmatrix}$$

Let us call relations and operations of algebra of sets with preliminary addition of missing attributes to NTA objects *generalized operations and relations* and denote them as follows: $\cap_G$, $\cup_G$, $\setminus_G$, $\subseteq_G$, $=_G$, etc. The first two operations completely correspond to logical operations $\wedge$ and $\vee$. NTA relation $\subseteq_G$ corresponds to *deducibility relation* in predicate calculus. Relation $=_G$ means that two structures are equal if they have been transformed to the same relation diagram by adding certain attributes. This technique offers a fundamentally new approach to constructing logical inference and deducibility checks introduced below; but first let us describe some examples of expressing conventional mathematical structures by means of NTA objects.

4 DATA AND KNOWLEDGE REPRESENTATION IN NTA

4.1 Graphs and Semantic Networks

In computers, graphs and networks are usually represented as list structures. In artificial intelligence systems, logical inference in graphs and semantic networks is implemented through algorithms of search for accessible vertices or through construction of the transitive closure of a graph. However, such algorithms are not efficient enough and hard to parallelize. Let us now consider the way graphs are expressed in NTA. We will use the graph presented in Figure 1 as an example.

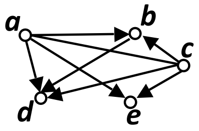

Fig. 1. Example of a graph

This graph can be expressed as a C-system $G[XY] = \begin{bmatrix} \{a\} & \{b,c,d,e\} \\ \{b\} & \{d\} \\ \{c\} & \{a,b,d,e\} \end{bmatrix}$ isomorphic to the adjacency matrix of this graph.

Composition of graphs $G \circ G$, e.g. composition of a graph with itself, is used quite often. This operation is shortly denoted as G^2. Greater "degrees" of composition can also be used, e.g. $G^3 = G \circ G \circ G$ and so on.

It is often necessary to determine the set of all the accessible vertices for each vertex of a graph G. This information is contained in the transitive closure of the graph, which is defined as follows.

Transitive closure of a graph G that contains n vertices, is the graph G^+ each of whose vertices is connected with all its accessible vertices with an arc.

Transitive closure can be constructed with the following sequence of operations:

$$G^+ = G \cup G^2 \cup G^3 \cup \ldots \cup G^k,$$

where $k \leq n$. Practically in all cases, the operation of transformation of a finite graph G into graph G^+ ends before the last "degree" G^k is found. The reason for ending this operation early is the fact that at some step the next "degree" of the graph does not have any arcs that have not been in the graph before.

Let us consider the way inference in semantic networks is implemented in NTA [15]. Any semantic network can be represented as a totality of binary relations. In semantic networks, inference rules are expressed as productions whose left part contains joins or compositions of some of these relations, and the right part is a relation that is substituted for the left part in the semantic network or is added to the semantic network as a new relation. Suppose that in an initial semantic network, existing relations R_1 and R_2 (see Figure 2) infers an additional link R_3

between the domain of the relation R_1 (vertex K) and the co-domain (target) of the relation R_2 (vertex N). The respective rule is shown in Figure 3 where A, B, C are variables whose values can be the vertices of the described semantic networks.

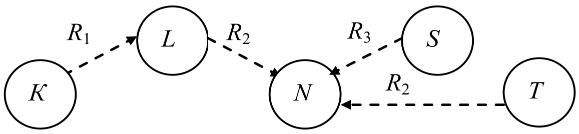

Fig. 2. Initial semantic network

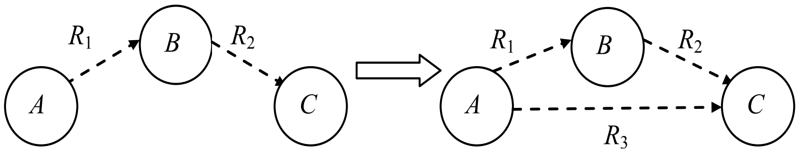

Fig. 3. Example of a transformation rule for a network

In NTA language, this network can be recorded as a totality of C-systems, namely $R_1[XY] = [\{K\}\{L\}]$, $R_2[YW] = [\{L, T\}\{N\}]$, $R_3[XW] = [\{S\}\{N\}]$.

To express a production rule by means of NTA in general case, we need to perform the following sequence of steps:

1. calculate the join of relations contained in the left part of the rule;
2. filter the resulting relation P using some the given restrictions, for instance, a totality of facts;
3. if the rule requires substituting its left part for the right one, delete all links contained in P from the initial knowledge base;
4. calculate the join T of relations contained in the right part of the rule;
5. filter T if necessary;
6. add all links contained in T into the knowledge base.

Classifying the rules in advance simplifies the calculations significantly. As the rule shown in Figure 3 only requires adding a new link without deleting any other links, we only need to calculate $R_1[XY] \circ R_2[YW] = [\{K\}\{L\}] \circ [\{L, T\}\{N\}] = [\{K\}\{N\}]$ and then add the derived n-tuple into the C-system matched to the relation $R_3 : R_3[XW] = R_3[XW] \cup (R_1[XY] \circ R_2[YW]) = [\{S\}\{N\}] \cup [\{K\}\{N\}] = [\{S, K\}\{N\}]$. After all the necessary transformations, the semantic network will look as follows: $R_1[XY] = [\{K\}\{L\}]$, $R_2[YW] = [\{L, T\}\{N\}]$, $R_3[XW] = [\{S, K\}\{N\}]$.

4.2 Correspondence between N-tuple Algebra and Predicate Calculus

In trivial case (when individual attributes do not correspond to multiplace relations), an n-tuple corresponds to conjunction of one-place predicates with different variables. For example, a C-n-tuple $P[XYZ] = [P_1P_2P_3]$ where $P_1 \subseteq X$; $P_2 \subseteq Y$; $P_3 \subseteq Z$ corresponds to a logical formula $H = P_1(x) \wedge P_2(y) \wedge P_3(z)$.

A D-n-tuple $\overline{P} = \left]\overline{P_1}\,\overline{P_2}\,\overline{P_3}\right[$ corresponds to the negation of the formula H (disjunction of one-place predicates) $\neg H = \neg P_1(x) \vee \neg P_2(y) \vee \neg P_3(z)$.

An elementary n-tuple that is a part of a non-empty NTA object corresponds to a satisfying substitution in a logical formula.

An empty NTA object corresponds to an identically false formula.

An NTA object that equals any particular universe corresponds to a valid formula, or a tautology.

A non-empty NTA object corresponds to a satisfiable formula.

In NTA, attribute domains can be any arbitrary sets that are not necessarily equal to each other. This means that NTA structures correspond to formulas of many-sorted predicate calculus.

Now let us consider quantifiers in NTA.

If a dummy attribute is added to a C-n-tuple or a C-system, the procedure corresponds to the derivation rule of predicate calculus called generalization rule. For example, if an NTA object $G[XZ] = \begin{bmatrix} \{a,c\} & * \\ \{a,c,d\} & \{b,c\} \end{bmatrix}$ corresponds to a formula $F(x,z)$ of predicate calculus, by adding a dummy attribute Y into this NTA object we get an NTA object $G_1[XYZ] = +Y(G[XZ]) = \begin{bmatrix} \{a,c\} & * & * \\ \{a,c,d\} & * & \{b,c\} \end{bmatrix}$ which corresponds to the formula $\forall y F(x,z)$ derived from the formula $F(x,z)$ according to generalization rule. This relation is obvious for C-n-tuples and C-systems, but needs to be proved for D-n-tuples and D-systems.

Theorem 13. Adding a new dummy attribute to a D-n-tuple or a D-system corresponds to the formula $\forall y(P)$.

Proof. Let a D-n-tuple $P[X_1X_2\ldots X_n] =]P_1P_2\ldots P_n[$ be given. If we add a dummy attribute Y to it, we get $Q[YX_1X_2\ldots X_n] =]\varnothing P_1P_2\ldots P_n[$. Transforming this NTA objects into C-systems, we have

$$P = \begin{bmatrix} P_1 & * & \ldots & * \\ * & P_2 & \ldots & * \\ \ldots & \ldots & \ldots & \ldots \\ * & * & \ldots & P_n \end{bmatrix}; \quad Q = \begin{bmatrix} * & P_1 & * & \ldots & * \\ * & * & P_2 & \ldots & * \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ * & * & * & \ldots & P_n \end{bmatrix}$$

Hence, $Q = +Y(P) = \forall y(P)$. □

Suppose that a D-system $R[X_1X_2\ldots X_n]$ is given. Let $R_1 = +Y(R) = R_1[YX_1 X_2\ldots X_n]$. In the D-system R_1, "$\varnothing$" are components of the attribute Y in all D-n-tuples. After transforming this D-system into a C-system according to Theorem 9, the results in projection $[X_1X_2\dots X_n]$ are the same as in transformation of the D-system R into a C-system, and dummy components "*" are now components of the attribute Y in the C-system R_1. Therefore, $R_1 = +Y(R) = \forall y(R)$.

The two theorems that follow define the semantics of the operation -Attr.

Theorem 14. Let $R[\dots X\dots]$ be a C-system that has no C-n-tuples with empty components in the X attribute. Then for the predicate $P(\dots, x, \dots)$ that corresponds to this C-system, the formula $-X(R)$ corresponds to the formula $\exists x(P)$.

Proof. Let R be a C-n-tuple. Then under the conditions of the theorem correspondence $-X(R) \Leftrightarrow \exists x(P)$ is evident. Let R be a C-system that contains C-n-tuples R_1, R_2, ..., R_n. This means that $R = R_1 \cup R_2 \cup \dots \cup R_n$. Formula $P = P_1 \vee P_2 \vee \dots \vee P_n$, where P_i are formulae that correspond to C-n-tuples R_i corresponds to this formula in predicate calculus. Applying $-X$ operation to R, we get $-X(R) = -X(R_1) \cup -X(R_2) \cup \dots - X(R_n)$.

A formula of predicate calculus $\exists x(P_1) \vee \exists x(P_2) \vee \dots \vee \exists x(P_n)$ corresponds to the right part of the above equality. According to the rules of equivalent transformations in mathematical logic, the formula equals to a formula $\exists x(P_1 \vee P_2 \vee \dots \vee P_n)$, which is $\exists x(P)$ after substitution. □

Theorem 15. Let $R[\dots X\dots]$ be a D-system that has no D-n-tuples with components "*" in the X attribute. Then for a predicate $P(\dots, x, \dots)$ corresponding to this D-system, formula $-X(R)$ corresponds to the formula $\forall x(P)$.

Proof. The formula $\forall x(P)$ is known to be equal to $\neg(\exists x(\neg P))$. A C-system $\overline{R}$ that equals to the complement of the D-system R corresponds to the expression $\neg P$. $Q = -X(\overline{R})$ corresponds to the formula $\exists x(\neg P)$ since satisfies the conditions of Theorem 14. Then $\neg(\exists x(\neg P))$ is an NTA object that equals a D-system all of whose components equal the complements of corresponding components of Q. Therefore, $\overline{Q} = -X(R)$ as the attribute X has been eliminated from the C-system $\overline{R}$. □

Hence, if an attribute (e.g. X) is eliminated from a C-system, it means that the quantifier $\exists x$ is applied to this object, and if this attribute is eliminated from a D-system, it means that the quantifier $\forall x$ is applied to this object. For example, let a C-system and its complement expressed as a D-system be given:

$$Q[XYZ] = \begin{bmatrix} \{a,b,d\} & \{f,h\} & \{b\} \\ \{b,c\} & * & \{a,c\} \end{bmatrix}$$

and

$$\overline{Q}[XYZ] = \left]\begin{matrix} \{c\} & \{g\} & \{a,c\} \\ \{a,d\} & \emptyset & \{b\} \end{matrix}\right[.$$

Then

$$\exists x(Q[XYZ]) = \begin{bmatrix} \{f,h\} & \{b\} \\ * & \{a,c\} \end{bmatrix}$$

and

$$\forall x(\overline{Q}[XYZ]) = \left] \begin{array}{cc} \{g\} & \{a,c\} \\ \emptyset & \{b\} \end{array} \right[.$$

The next section is concerned with logical inference techniques in NTA. These techniques are applicable to the knowledge and data structures introduced earlier, as well as to certain different ones (e.g. relational tables in deductive DBs).

4.3 Systems with Measurable Attributes

Many attributes, e.g. duration, length, etc., can be given as systems of open, closed and semi-open intervals. Modern measure theory is based on this type of data being described by semi-ring algebra [8], while in NTA algebra of components corresponds to the laws of algebra of sets. As we found, this incompliance can be eliminated with the following method that we called interval quantization method (IQM).

Let a closed interval Ω on a numeric axis be the definitional domain of a certain attribute, and a finite set $E = \{E_i\}$ of closed intervals be given for which $E_i \subseteq \Omega$. On the numeric axis, the margins of the intervals are represented by sets of coordinates of their initial and end points. By arranging these coordinates in ascending order, we can split the system of intervals into quanta, i.e. into points and open intervals. It is clear that in this case the interval Ω is split into a certain composite set that contains m open intervals and $m + 1$ isolated points, two of which are endpoints of the interval Ω. Methodological difficulties caused by the fact that the set consists of heterogeneous objects (points and intervals) can be solved if we define a point as a degenerate interval of zero measure.

An example of quantization process for four intervals E_1, E_2, E_3, E_4 is shown in Figure 4. Intervals E_i are moved above the numeric axis for visualization purposes.

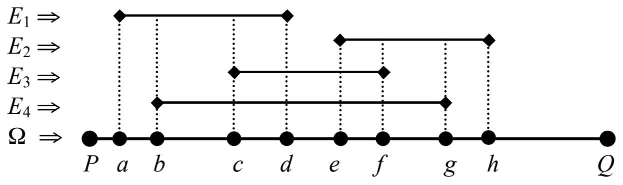

Fig. 4. Quantization of an interval system

Here, the interval Ω whose endpoints are P and Q, contains inner points a, b, $\dots$, h. Accordingly, each interval E_i can be represented as a set of quanta; for example, the closed interval E_3 is a set that contains points and open intervals: $E_3 = \{c, d, e, f, (c, d), (d, e), (e, f)\}$. If we are concerned only with the metric properties of the objects that we are describing, we can represent the interval E_3 as a set $\{(c, d), (d, e), (e, f)\}$ of open non-intersecting intervals. By similar quantization for each measurable attribute, we can represent the metric space as an NTA object whose components are ordinary sets, the introduced notations for points and intervals being their elements. Methods of immersion NTA structures into a metric space are now used for logic-probabilistic analysis of systems [14].

When an interval system (Ω, E) for a certain attribute is transformed into an elementary interval system (Ω, F), where $E = \{E_i\}$ is a set of the initial intervals, and $F = \{F_r\}$ is a set of quanta, the following relation is true for any E_i:

$$E_i = \bigcup_k F_k,$$

where F_k are certain quanta of F. In this case, the measure μ for the initial interval E_i is calculated by the formula below:

$$\mu(E_i) = \sum_k \mu(F_k).$$

We have proved that in NTA, if each component of a C-n-tuple has a finite measure, the measure of this C-n-tuple is the product of its components' measures. For example, for a C-n-tuple $C_1 = [\{(a,c),(e,g)\}\,\{(i,k),(n,p)\}]$,

$$\mu(C_1) = (\mu((a,c)) + \mu((e,g))) \times (\mu((i,k)) + \mu((n,p))).$$

When calculating the measure of a C-system it is important to remember that the intersection of its C-n-tuples can be nonempty, and to make applicable corrections using the relation $\mu(A \cup B) = \mu(A) + \mu(B) - \mu(A \cap B)$ for arbitrary C-n-tuples A and B.

Now let us consider an example of IQM implementation. Let the following logical formula whose satisfiability needs to be checked, be given:

$$(x > 3) \wedge (x < 4) \wedge (y > 2) \wedge (y < 7) \wedge (z > 5) \wedge (z < 6) \wedge (x > y) \wedge (y > z). \quad (3)$$

Considering the applicable generalized operations (see Section 3.2) the NTA expression below corresponds to the analyzed formula:

$$P[XYZ] \cap_G MORE[YZ] \cap_G MORE[XY],$$

where the C-n-tuple $P[XYZ] = [\{(3,4)\},\{(2,7)\},\{(5,6)\}]$ corresponds to the expression $(x > 3) \wedge (x < 4) \wedge (y > 2) \wedge (y < 7) \wedge (z > 5) \wedge (z < 6)$. Relations $MORE[YZ]$ and $MORE[XY]$ correspond to the predicates $(x > y)$ and $(y > z)$, their generalized intersection corresponding to the relational join operation. Thus, the satisfiability problem comes down to finding the measure of a C-system $P[XYZ] \cap_G (MORE[YZ] \oplus MORE[XY])$. If this measure is not equal to zero, the formula (3) is satisfiable, a domain of nonzero volume corresponding to this formula in property space $X \times Y \times Z$. If the opposite is true, the formula is not satisfiable.

Since only the metric aspects are of interest to us, let us express the components of the C-n-tuple $P[XYZ]$ trough quanta $(2,3)$, $(3,4)$, $(4,5)$, $(5,6)$, $(6,7)$:

$$P[XYZ] = [\{(3,4)\},\{(2,3)\},\{(3,4)\},\{(4,5)\},\{(5,6)\},\{(6,7)\},\{(5,6)\}]\,.$$

Having substituted the values of attributes from the $P[XYZ]$ into predicates $(x > y)$ and $(y > z)$, we get sets of points that comprise C-systems $MORE[XY]$ and $MORE[YZ]$ (these points are shown in dark grey in Figure 5):

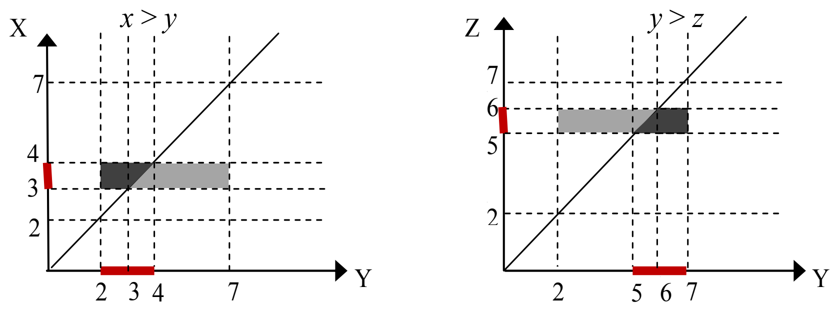

Fig. 5. Cartesian products of intervals

Obviously, these two areas have no common elements (quanta) in the attribute Y. Therefore, $MORE[YZ] \oplus MORE[XY] = \varnothing$, and $\mu(MORE[YZ] \oplus MORE[XY]) = 0$.

Thus, the IQM allows to determine unsatisfiability of logical formulae which contain measurable attributes as well as implement logical inference based on structure analysis of logical formulae, including formulas that contain elementary unitary and binary predicates with no quantifiers [21].

4.4 Relational Database Management Systems

Relations using the primary key concept are a particular case of NTA objects, since any NTA object can be split into a set of elementary n-tuples. However, elementary n-tuples do not use specific properties of NTA structures, thus it is rational to use NTA only for relations whose n-tuple components are sets, not elements. Such relations can be used for representing graphs and networks, as well as some projections of regular DB tables. If required, associative search can be an efficient alternative to primary key search method in such structures.

Let us consider the way DBMS queries are expressed in NTA. Let a BD use a relation expressed as an NTA object $P[XYZ]$. In the relation P, we need to find all possible values for attributes X and Y, attribute Z being within the given range D. In SQL, this query looks as follows:

$$\text{SELECT } X, Y \text{ FROM } P \text{ WHERE } Z \subseteq D.$$

In NTA, this query is expressed through an NTA object called a selector, in this case, a C-n-tuple $Q_1[Z] = [D]$. We can get the answer to the query by calculating $P[XYZ] \cap_G Q_1[Z]$.

Let us consider an example in which a relation join is required. Suppose that, beside the relation $P[XYZ]$, our DB contains a relation $R[YVW]$, and we need to find the values X and V, if $Z = a$. In SQL, this is written as SELECT X, V FROM P, R WHERE $Z = a$ AND $P.Y = R.Y$.

Obviously, in NTA the relation diagram of the query corresponds to the relation diagram of the NTA object derived by joining P and R. Then the query can be written as a C-n-tuple $Q_2[Z] = [\{a\}]$, and the answer to the query are attributes X and V, as calculated by this formula:

$$(P[XYZ] \oplus R[YVW]) \cap_G Q_2[Z].$$

NTA allows implementing queries that are impossible in DBMS, such as queries addressed to relation complements. This can be implemented not only through C-n-tuples and C-systems, but also through more complex NTA objects.

In NTA structures, recursive queries can be implemented through calculating transitive closures of the corresponding relations, followed by selecting and eliminating attributes. This subject is discussed in detail in the section below.

4.5 Deductive Databases

Deductive DBMS widely use functional systems theory and proof-theoretic approach. In such DBMS, query execution involves proving a certain theorem through special deductive axioms and inference rules. Here, the basic axioms corresponding to domain elements and n-tuples of basic relations constitute the extensional database (EDB) of the DBMS, and the auxiliary axioms and consistency constraints comprise its intensional database (IDB). A language of any calculus used in formulating query and in logical inference for answering it, is commonly called a Datalog, and a description in this language is called a Datalog program. The term "Datalog rules" refers to the part of the Datalog program that contains no facts. One of the features of deductive DBMS is recursive query support. Let us consider an example of a Datalog program in which the facts are arranged in the Points table (see Table 1), and the rules allow finding all pairs of values of A (departure point) and B (destination point), where A and B are, respectively, the beginning and the end of a valid route from A to B.

Departure point	*Destination point*
Washington	Los Angeles
Los Angeles	New York
New York	Washington
New York	Chicago

Table 1. Points

Datalog rules:

$$\begin{gathered} \mathit{points}(\mathit{departure\_point}; \mathit{destination\_point}) \\ \supset \mathit{route}(\mathit{departure\_point}, \mathit{destination\_point}), \end{gathered}$$

$$\begin{gathered} \mathit{route}(\mathit{departure\_point}, \mathit{intermediate\_point}) \\ \wedge\ \mathit{route}(\mathit{intermediate\_point}, \mathit{destination\_point}) \\ \wedge\ \mathit{departure\_point} \neq \mathit{destination\_point} \supset \mathit{route}(\mathit{departure\_point}, \mathit{destination\_point}). \end{gathered}$$

In NTA language, the predicate (relation) Points can be represented as follows:

$$T = \begin{bmatrix} \{\text{Washington}\} & \{\text{LosAngeles}\} \\ \{\text{LosAngeles}\} & \{\text{NewYork}\} \\ \{\text{NewYork}\} & \{\text{Washington, Chicago}\} \end{bmatrix}.$$

The given Datalog program is implemented in NTA through a transitive closure $T^+ = T \cup T^2 \cup T^3 \cup \ldots \cup T^k$, where $k \leq n$. This can be obtained in several steps.

Step 1.

$$\begin{aligned} T^2 = T \circ T &= \begin{bmatrix} \{\text{Washington}\} & \{\text{NewYork}\} \\ \{\text{LosAngeles}\} & \{\text{Washington, Chicago}\} \\ \{\text{NewYork}\} & \{\text{LosAngeles}\} \end{bmatrix}, \\ T \cup T^2 &= \begin{bmatrix} \{\text{Washington}\} & \{\text{LosAngeles, NewYork}\} \\ \{\text{LosAngeles}\} & \{\text{NewYork, Washington, Chicago}\} \\ \{\text{NewYork}\} & \{\text{Washington, Chicago, LosAngeles}\} \end{bmatrix}. \end{aligned}$$

Evidently, at the first step, the C-system derived from union of T and T^2 contains new elementary n-tuples, as compared to the C-system T. Now let us go on to the second step.

Step 2.

$$T^3 = \begin{bmatrix} \{\text{Washington}\} & \{\text{Washington, Chicago}\} \\ \{\text{LosAngeles}\} & \{\text{LosAngeles}\} \\ \{\text{NewYork}\} & \{\text{NewYork}\} \end{bmatrix},$$

Since the departure point is not equal to the destination point, the final line is $T^3 = \begin{bmatrix} \{\text{Washington}\} & \{\text{Chicago}\} \end{bmatrix}$.

$$T \cup T^2 \cup T^3 = \begin{bmatrix} \{\text{Washington}\} & \{\text{LosAngeles, NewYork, Chicago}\} \\ \{\text{LosAngeles}\} & \{\text{NewYork, Washington, Chicago}\} \\ \{\text{NewYork}\} & \{\text{Chicago, Washington, LosAngeles}\} \end{bmatrix},$$

Executing the rest of the steps yields no changes in the final table; therefore, we have obtained the transitive closure T^+ of the relation T. The relation T^+ can be matched to the Route predicate in the Datalog rules.

5 LOGICAL INFERENCE IN NTA

5.1 Computational Complexity of Algebraic Operations in Logical Inference

The most popular systems of logical inference in mathematical logic are as follows:

1. Hilbert-style calculi proposed in [7];
2. natural deduction calculus developed by logician G.Gentzen [6];
3. logical inference based on Resolution Principle that became widely known after the article [2] was published.

Logical inference systems often use two theorems introduced and proved in [1] (they have numbers 2.1 and 2.2 there). They are reproduced below since they allow to derive logical corollaries by algebraic methods as well as by inference rules.

Theorem 16. Let formulas $F_1, \dots, F_n$ and G be given. Then G is a logical corollary to $F_1, \dots, F_n$ if and only if the formula $((F_1 \wedge \dots \wedge F_n) \supset G)$ is a valid one.

Theorem 17. Let formulas $F_1, \dots, F_n$ and G be given. Then G is a logical corollary to $F_1, \dots, F_n$ if and only if the formula $(F_1 \wedge \dots \wedge F_n \wedge \neg G)$ is inconsistent.

Logical inference in NTA is based on the Theorems 16 and 17 which can be expressed in NTA terms as follows, since NTA is isomorphic to algebra of sets.

Method 1. Let NTA objects $F_1, \dots, F_n$ and G be given. Then G is a logical corollary to $F_1, \dots, F_n$ if and only if $(F_1 \cap_G \dots \cap_G F_n) \neq \emptyset$ and $(F_1 \cap_G \dots \cap_G F_n) \subseteq_G G$.

Method 2. Let NTA objects $F_1, \dots, F_n$ and G be given. Then G is a logical corollary to $F_1, \dots, F_n$ if and only if $(F_1 \cap_G \dots \cap_G F_n) \neq \emptyset$ and $F_1 \cap_G \dots \cap_G F_n \cap_G \overline{G} = \emptyset$.

NTA structures can be polynomially reduced to logical ones; hence computational complexity of algorithms on NTA structures fully corresponds to computational complexity of algorithms solving problems on logical structures. A significant number of such problems arising during logical analysis by means of deduction procedures, for instance, the satisfiability problem for a conjunctive normal form (CNF), are NP-complete problems with regard to their computational complexity (i.e. they require algorithms of exponential complexity). However, there are many special cases that are solvable in polynomial time only. As far as the problem of CNF satisfiability is concerned, they are CNFs with at most two literals in every clause or CNFs with Horn clauses only. Identifying cases where we can recognize satisfiability in polynomial time is of great importance for applied research since it reduces the time required for implementation of algorithms.

The special cases mentioned above can be expressed in NTA structures as well; however, NTA has its own means for reducing laboriousness and sometimes computational complexity of algorithms. These will be briefly introduced in the next section.

In NTA, deducibility checks are not based on inference rules; rather, they check enclosure of certain NTA objects into each other or emptiness of intersection of certain relations including NTA objects related to alternative classes. Consequently, in order to implement logical inference in NTA, we need to solve two key problems, namely enclosure check for two NTA objects and transformation of an NTA object into one of alternative classes. In general case, complexity of these problems is greater than polynomial, coinciding with complexity of similar problems expressed in terms of mathematical logic.

Enclosure check for two NTA objects ($A \subseteq_G B$) corresponds to validity check for implication $A \supset B$ in logic. Transformation of an NTA object into object of an alternative class is equivalent to transformation a CNF into a DNF or vice versa.

Table 2 contains different combinations of NTA objects; those marked with a symbol "+" are the ones for which the algorithms for applicable operations are polynomial, given that all attribute domains are sets rather than multiplace relations.

Operation	*C-n-tuple*	*C-system*	*D-n-tuple*	*D-system*
Member check for an elementary n-tuple	+	+	+	+
Check of enclosure of a C-n-tuple into	+		+	+
Check of enclosure of a C-system into	+		+	+
Check of enclosure of a D-n-tuple into	+		+	+
Check of enclosure of a D-system into				

Table 2. Complexity of NTA operations

Table 2 shows that computational complexity of operations depends on the structure class of the NTA objects used in the operations. For instance, enclosure check of a C-n-tuple into a C-system has exponential computational complexity while enclosure check of a C-n-tuple or even a C-system into a D-system is polynomial.

Transformation of an NTA object into one of an alternative class when the initial object is a D-system or a C-system is computationally harder than an NP-complete problem; it belongs to the class of #P-complete problems, i.e. enumeration ones.

Maximum complexity estimate for transformation of an NTA object into one of an alternative class is easy to calculate. Suppose we have a D-system of dimension $M \times N$, where M is the number of rows and N is the number of columns of this structure. Then every D-n-tuple can be transformed into a C-system with no more than N rows, and solving this problem will take $M - 1$ sequential calculations of intersections. If we consider the complexity of intersection for two C-n-tuples a constant B, then the maximum computational complexity of this operation is

$B \cdot N^{M-1}$ given that every intersection is nonempty. This estimate is evidently greater than the one for a problem of CNF satisfiability.

Transformation of D-systems into C-systems is of most interest for implementation of logical inference, since in NTA a D-system is isomorphic to a CNF and a C-system can be considered a set of satisfying substitutions for this CNF. Hence for this transformation we need to solve a problem of CNF satisfiability, one of the most popular problems in applications of logical inference theory and computational complexity theory. This solution is clearly redundant as it gives all elementary n-tuples or C-tuples contained in the D-system, while it would be enough to find only one of them to declare the D-system is not empty; but the solution is acceptable when some analysis of satisfying substitutions is needed beside determining of the CNF satisfiability.

In practice, a CNF satisfiability (D-system emptiness) check is the initial stage of any algorithm for transformation of a D-system into a C-system since the transformation makes sense only if the initial D-system is not empty. This stage can be the only one if no analysis of satisfying substitutions is required. A CNF satisfiability check is also performed during enclosure checks for NTA objects; moreover, it is the main source of complexity in this case.

The material stated above lets us conclude that in NTA, as well as in many other logical systems, a laboriousness decrease for the problem of CNF satisfiability allows to shorten not only certain steps of logical inference algorithms, but also the whole inference procedure, if it can be reduced to the said problem.

In NTA, a decrease in laboriousness and sometimes in computational complexity as well, is mostly realized by using matrix properties of NTA objects; we are now going to describe these properties below.

5.2 Matrix Properties of NTA Objects (Case Study: CNF Satisfiability)

In artificial intelligence systems, CNF satisfiability is an important problem for the following reasons:

1. In complexity theory, CNF satisfiability is the basic NP-complete problem. It has been proven that, using rational coding, any NP-class problem can be represented as a CNF, and that converting any NP-class problem to CNF takes polynomial time.
2. Based on Theorem 17, the resolution principle was stated for both propositional and predicate calculus; this principle is now widely used in machine implementations of artificial intelligence systems. According to this principle, the formula $F_1 \wedge \ldots \wedge F_n \wedge \neg G$ is converted to CNF, and logical inference is narrowed down to solving a CNF satisfiability problem.

Thus, it is important to find new CNF classes with a polynomially recognizable satisfiability property.

NTA structures (n-tuples and systems of n-tuples) look like matrices. Although operations on NTA objects are substantially different from matrix-algebra operations, the former have many properties of matrix structures, which can be used to reduce laboriousness in many NTA operations, e.g. in solving the problem of CNF satisfiability/emptiness of a D-system.

Let an arbitrary D-n-tuple $=]s_1 s_2 \dots s_n[$ be given. If the set of its components can be divided into R nonempty nonintersecting subsets that make up D-n-tuples D_j, then

$$D = \bigcup_{j=1}^{R} D_j.$$

For example, if a D-n-tuple $]s_1 s_2 s_3 s_4 s_5 s_6 s_7[$ is split into three D-n-tuples $]s_2 s_4 s_6[$, $]s_1 s_3 s_5[$ and $]s_7[$, reducing these to the same relation diagram transforms them into D-n-tuples $]\emptyset s_2 \emptyset s_4 \emptyset s_6 \emptyset[$, $]s_1 \emptyset s_3 \emptyset s_5 \emptyset\emptyset[$ and $]\emptyset\emptyset\emptyset\emptyset\emptyset\emptyset s_7[$. Union of these relations equals the initial D-n-tuple.

Similarly, under the same splitting conditions, for C-n-tuples C, C_i it is true that

$$C = \bigcap_{j=1}^{R} C_j.$$

Let T be a D-system represented as a matrix of components whose dimensions are $M \times N$ (M rows and N columns). Let us divide this D-system into R vertical blocks R ($j = 1, 2, \dots, R$) by vertical lines. Let D_i ($i = 1, 2, \dots, M$) be a D-n-tuple that is represented by the i^{th} row of the matrix T, then $T = \bigcap_{i=1}^{M} D_i$. Let D_{ij} denote the D-n-tuple represented by a subrow of the i^{th} row from the j^{th} block in the matrix T. Then, $D_i = \bigcup_{j=1}^{R} D_{ij}$, and $T = \bigcap_{i=1}^{M} \left(\bigcup_{j=1}^{R} D_{ij} \right)$. By removing the brackets in the last equation, we get the relation stated in the following theorem.

Theorem 18. If a D-system matrix T with dimensions $M \times N$ is divided into R vertical blocks ($R < N$), the following is true:

$$T = \bigcup_{j=1}^{S} \left(\bigcap_{i=1}^{M} D_{ij} \right),$$

where $S = R^M$.

One of the corollaries to the Theorem 18 is that a D-system is nonempty if at least one D-system $\bigcap_{i=1}^{M} D_{ij}$ is nonempty, whatever the division into vertical blocks is. For example, D-system $\left] \begin{array}{c|ccc} t_{11} & t_{12} & t_{13} & t_{14} \\ t_{21} & t_{22} & t_{23} & t_{24} \end{array} \right[$ is divided into two vertical blocks. In accordance with the theorem, it can be represented as a union of four D-systems

$$\left(\left] \; t_{11} \quad \emptyset \quad \emptyset \quad \emptyset \; \right[\cup \left] \; \emptyset \quad t_{12} \quad t_{13} \quad t_{14} \; \right[\right)$$

$$\cap \left(\left] \; t_{21} \quad \emptyset \quad \emptyset \quad \emptyset \; \right[\cup \left] \; \emptyset \quad t_{22} \quad t_{23} \quad t_{24} \; \right[\right)$$

$$= \left] \begin{matrix} t_{11} & \emptyset & \emptyset & \emptyset \\ t_{21} & \emptyset & \emptyset & \emptyset \end{matrix} \right[\cup \left] \begin{matrix} t_{11} & \emptyset & \emptyset & \emptyset \\ \emptyset & t_{22} & t_{23} & t_{24} \end{matrix} \right[$$

$$\cup \left] \begin{matrix} \emptyset & t_{12} & t_{13} & t_{14} \\ t_{21} & \emptyset & \emptyset & \emptyset \end{matrix} \right[\cup \left] \begin{matrix} \emptyset & t_{12} & t_{13} & t_{14} \\ \emptyset & t_{22} & t_{23} & t_{24} \end{matrix} \right[.$$

If at least one of the final D-systems is nonempty, then the initial D-system is nonempty as well.

While being inapplicable to solving practical problems, this theorem provides the grounds for some corollaries of practical importance. Let us introduce some new terms that we use for considering these corollaries below.

A *conflicting pair* of a D-system is a nonintersecting pair of nonempty components in the same attribute.

A *conflict attribute* of a D-system is an attribute in which the intersection of all nonempty components is empty; if this intersection is nonempty, this attribute is a non-conflict attribute.

A *non-conflict block* (*BlNC*) of a D-system is its vertical block that contains only non-conflict attributes. Respectively, a vertical block that contains conflict attributes is a conflict block (*BlC*). From the definitions, it is clear that a non-conflict block can contain empty components, while the intersection of all nonempty components in each attribute of this block is nonempty.

A *monotonous attribute* of a D-system is a non-conflict attribute that contains no empty components.

A *monotonous block* (*BlM*) of a D-system is a non-conflict block that contains no D-n-tuples all of whose components are empty.

For a D-system $Q[X]$ containing a monotonous block $BlM(Q)$, let us construct a C-n-tuple $C_{\text{int}}[X]$, each of whose components that corresponds to an attribute from the relation diagram of the monotonous block equals the intersection of all nonempty components of the block belonging to this attribute, the rest of the components of this C-n-tuple being dummy ones, i.e. equal to "*".

Theorem 19. If a D-system Q contains a monotonous block, it is nonempty, and $C_{\text{int}} \subseteq Q$.

Proof. It is clear that under the hypothesis of the theorem $C_{\text{int}} \neq \emptyset$, and that for each D-n-tuple Q_i in the system Q $C_{\text{int}} \subseteq Q_i$ is true, since each monotonous block always has a nonempty component which includes the corresponding C_{int} component. This implies that $C_{\text{int}} \subseteq Q$. $\square$

Let us consider the following D-system as an example:

$$T = \left] \begin{matrix} \{A, B\} & \{f, g\} & \{a, c, d\} \\ \emptyset & \{e\} & \{b, c\} \\ \{A\} & \{g, h\} & \emptyset \end{matrix} \right[$$

It is easy to see that the first and the third attribute in the T are non-conflict and that they make up a monotonous block. Respectively, $C_{\text{int}} = [\{A\} * \{c\}]$.

According to Theorem 19, T is nonempty and $C_{\text{int}} \subseteq T$, which is also verified through Theorems 11 and 12.

Let us now consider D-systems with non-conflict blocks that contain D-n-tuples all of whose components are empty. Let a D-system Q contain a non-conflict block $T_1 = BlNC(Q)$. Then after the applicable swaps of rows and columns, the D-system Q can be represented as a block matrix $T = \left\| \begin{matrix} T_{11} & T_{12} \\ T_{21} & T_{22} \end{matrix} \right\|$, where T_{11} is a submatrix of the matrix Q whose D-n-tuples have no empty components and T_{12} is a submatrix of the matrix Q whose D-n-tuples contain only empty components.

Theorem 20. If a D-system Q has a non-conflict block, then Q is nonempty, if and only if after dividing Q into a block matrix T the D-system represented by the block T_{22} is non-empty.

Proof. Necessity. From Theorem 18, it is clear that one of the D-systems that is the result of intersection of the blocks on the main diagonal can be represented as a block matrix $\left\| \begin{matrix} T_{11} & \emptyset \\ \emptyset & T_{22} \end{matrix} \right\|$. Since the D-system T_{11} is monotonous, there is a nonempty C-n-tuple C_{11}, where $C_{11} \subseteq T_{11}$. If T_{22} is nonempty, in the projection corresponding to T_{22} there exists such a C-n-tuple C_{22} that $C_{22} \subseteq T_{22}$. Intersection of C_{11} and C_{22} forms a non-empty C-n-tuple C_0, since all non-dummy components of the n-tuple C_{11} correspond to the dummy components of the C-n-tuple C_{22} and vice versa. From Theorems 11 and 12, $C_0 \subseteq T$. Sufficiency. Suppose that T_{22} is empty, and Q is nonempty. Then T_{22} is equivalent to a D-system of the same dimension, all of whose components are empty sets. If we substitute this D-system for T_{22}, the lower part of the block matrix will contain D-n-tuples all of whose components are empty; therefore, the corresponding D-system is also empty. □

Let us analyze the following D-system as an example:

$$Q = \left] \begin{matrix} \{A, B\} & \{e, f\} & \{a, b\} & \emptyset \\ \emptyset & \{g, h\} & \emptyset & \{e\} \\ \{A\} & \{e\} & \{b\} & \{f, g\} \\ \emptyset & \{e, h\} & \emptyset & \{g, h\} \end{matrix} \right[.$$

Its first and third columns are non-conflict. Therefore, after the applicable swap of rows and columns, it can be converted to an equivalent block matrix

$$Q_1 = \left] \begin{array}{cc|cc} \{A, B\} & \{a, b\} & \{e, f\} & \emptyset \\ \{A\} & \{b\} & \{e\} & \{f, g\} \\ \hline \emptyset & \emptyset & \{g, h\} & \{e\} \\ \emptyset & \emptyset & \{e, h\} & \{g, h\} \end{array} \right[.$$

To check nonemptiness of this D-system it is sufficient to check nonemptiness of the D-system $T_{22} = \left] \begin{matrix} \{g, h\} & \{e\} \\ \{e, h\} & \{g, h\} \end{matrix} \right[$. The first column of this D-system is monotonous, therefore, it is nonempty and contains the C-n-tuple $C_{22} = [\{h\}*]$. The monotonous block T_{11} contains a C-n-tuple $[\{A\}\{b\}]$. After reducing these C-n-tuples to the same relation diagram, their intersection equals the C-n-tuple $C_0 = [\{A\}\{b\}\{h\}*]$. After swapping the attributes in accordance with the relation diagram of the initial D-system Q, this C-n-tuple is transformed into the C-n-tuple $[\{A\}\{h\}\{b\}*]$, which is the substitution for the D-system Q. With Theorems 11 and 12, we can verify that this substitution is correct.

This example shows that, in the same D-system, relations stated in Theorems 19 and 20 can be used more than once, and that in some cases solving a problem in polynomial time is possible. Obviously, for an arbitrary D-system, the algorithm for finding non-conflict and monotonous blocks is polynomially dependent on the dimensions of this system. Furthermore, if a D-system has any monotonous blocks, its nonemptiness is checked through an algorithm that is polynomially dependent on the matrix dimensions of this D-system; and if a D-system has any non-conflict blocks, another D-system of smaller dimensions can be used to check nonemptiness of the initial D-system.

In algorithms for solving CNF satisfiability problems and enclosure checks for NTA objects, such as for C-n-tuples in C-systems, matrix properties of NTA objects are widely used. At present, satisfiability recognition algorithms are developed to the greatest extent in propositional calculus. It has been proven [10] that, if a CNF is represented as a D-system that contains only three equally distributed (with the probability of 1/3) symbols: 1, 0 and $\emptyset$, this CNF is solvable, on the average, in polynomial time, this time being less than or equal to a third degree polynomial to the number of conjuncts.

5.3 New Features of Logical Inference In NTA

Previous sections were concerned with using NTA structures for implementing known methods of logical inference. New implementations of logical procedures based on the suggested algebraic approach are presented below.

Suppose that we have a system of axioms $A_1, \dots, A_n$ represented as NTA objects. Let us describe methods for solving the following two problems through NTA.

1. Problem of correctness check for a consequence. If we have an alleged consequence B, the proof procedure is a correctness check for the following generalized inclusion:
$$(A_1 \cap_G \dots \cap_G A_n) \subseteq_G B. \tag{4}$$
This relation allows correctness checks not only for the inference rules of classical logic, but also for rules specific to a certain knowledge system.
2. Problem of derivation of arbitrary consequences. In order to solve this problem, we first calculate an NTA object $A = A_1 \cap_G \dots \cap_G A_n$, after which we choose the B_i for which $A \subseteq_G B_i$ is true. The authors have developed algorithms that allow to calculate possible corollaries for a known A using the relation (4).

Below we will consider A as a C-system. If it is not true for a given A, it can be transformed into a system using algorithms for transforming D-n-tuples or D-systems into C-systems.

The following premises are commonly used for searching for possible consequences:

1. consequence Bi should preferably use only a small number of variables from the axioms $A_1, \ldots, A_n$;
2. the variables used are often determined based on semantic analysis of the given reasoning system.

Let us consider formal methods (i.e. without taking semantic restrictions into account) for solving Problem 2.

Decreasing the number of variables in B_i is possible through eliminating some attributes from A. Obviously, after this the transformation relation $A \subseteq_G B_i$ is true. Eliminating attributes from a C-system yields a projection whose properties determine the subsequent operations for consequence derivation. Such projection can be complete, i.e. can contain all elementary n-tuples for their relation diagram, or incomplete, if the opposite is true. If a projection is complete, it means that the consequence is a tautology and thus holds no interest for us; this is why we will consider incomplete projections only.

Let us form a group of incomplete projections for the A. In this case, all the variety of ways to form possible consequences B_i can be expressed by the following three rules:

1. keep one of the incomplete projections as a B_i;
2. choose any projection as a B_i, provided that it includes at least one incomplete projection;
3. for the NTA object chosen according to the rules above, construct, by adding elementary n-tuples or C-n-tuples, an incomplete NTA object that covers it.

   As an example, let us prove correctness of one of the inference rules in natural calculus called the dilemma rule:

$$\frac{A \to C, B \to C, A \vee B}{C}.$$

It is implied that the formulas below the solidus are derived from the ones above it. The upper formulas can be considered axioms, and the lower ones can be considered corollaries to these axioms. By transforming the conjunction of the formulas above the solidus into a D-system within the $[ABC]$ relation diagram, we get $Up[ABC] = \left]\begin{matrix} \{0\} & \emptyset & \{1\} \\ \emptyset & \{0\} & \{1\} \\ \{1\} & \{1\} & \emptyset \end{matrix}\right[$. The lower part of the rule can be expressed as a C-n-tuple $Dn[C] = [\{1\}]$. In order to prove by NTA methods that the given rule is true, we need to verify the relation $Up[ABC] \subseteq_G Dn[C]$. In this case, when the Up is a D-system and the inclusion check does not come down to algorithms of polynomial difficulty (see Table 2), it is rational to calculate $Up[ABC] \cap_G \overline{Dn[C]}$ and to check if the derived system is empty by using methods from the Section 5.2 and from [9].

Transforming $Up[ABC]$ into a C-system (this operation is often more laborious; then emptiness check for a D-system) yields $Up[ABC] = \begin{bmatrix} \{1\} & \{0\} & \{1\} \\ * & \{1\} & \{1\} \end{bmatrix}$. In this case, the inclusion check $Up[ABC] \subseteq_G Dn[C]$ is feasible by an algorithm of polynomial hardness.

This problem is a good example for implementing search for arbitrary consequences. Let us find incomplete projections in the C-system $Up[ABC]$. These projections are $[C]$, $[AB]$, $[AC]$ and $[BC]$. For the first projection, we get $Up[C] = \begin{bmatrix} \{1\} \\ \{1\} \end{bmatrix} = [\{1\}]$, which corresponds to the logical formula of the C. The projections $[AC]$ and $[BC]$ ultimately yield the same result. The projection $[AB]$ corresponds to the formula $A \vee B$.

The suggested approach allows to use algebraic methods for solving problems of logical inference. Moreover, it allows to see the essence of logical inference in classical logic in a new light. We know that if $A \subseteq B$ is true, it means that B is a necessary condition or a property of A. The relation (4) shows that a logical consequence is correct not only because it has been obtained using inference rules whose meaning may not always be clear, but also because it is a necessary condition for existence of the antecedent.

Above, we have already considered some examples of using algebraic approach for processing basic structures of data and knowledge. Let us now analyze in detail some ways of using NTA in certain existing program systems.

6 CONCLUSION

This article suggests using algebraic approach based on general theory of multiplace relations for solving logical analysis problems, the mathematical base for this approach being NTA, which is considered a Boolean structure in abstract algebra. The suggested generalized operations and relations significantly broadens the analytical scope and application field of NTA objects as compared to those of mathematical structures currently used for modeling and analyzing relations, e.g. in theory of binary relations or in relational algebra.

The research data given above shows that NTA allows to unify processing various data and knowledge structures in artificial intelligence systems. Todays knowledge representation languages are declarative, which makes it difficult to find efficient algorithms for information systems that use heterogeneous structures, as well as for assessing operation speed of an algorithm. Conversely, in n-tuple algebra, many declarative commands can be represented as relatively simple procedures. As for implementing logical inference procedures in n-tuple algebra, these can include, beside the known logical calculus methods, new algebraic methods for checking correctness of a consequence or for finding corollaries to a given axiom system.

In some cases, NTA provides a faster solution to standard logical analysis tasks as it considers not only feasibility of certain substitutions, but also the inner structure of knowledge to be processed. NTA allows to efficiently parallelize logical inference algorithms, i.e. to process knowledge in a way similar to that of tabular data processing in relational DBMS. Matrix properties of NTA objects allow to further decrease laboriousness of intellectual procedures. We found new structural and statistical classes of CNF with polynomially identifiable satisfiability properties in NTA. Consequently, many algorithms whose complexity evaluation is theoretically high, e.g. exponential, can in practice be solved in polynomial time, on the average. This substantiates that using algebraic approach is practical not only for data management, but also for knowledge processing.

We are planning to conduct future research in the following directions:

- context-oriented database (knowledge base) management systems [3, 20];
- research on additional means of immersing NTA structures into measure spaces [14];
- modelling intelligent dynamic systems [12, 18] within situational approach [4, 5].

Acknowledgment

The authors would like to thank the Russian Foundation for Basic Research (grants 12-07-00302, 11-08-00641, 12-07-00550, 12-07-00689), the Department for Nanotechnologies and Information Technologies of RAS (project 2.11 of the current Programme of Basic Scientific Researches), and the Chair of RAS (project 4.3 of the Program #16) for their help in partial funding of this research.

REFERENCES

[1] Chang, C.-L.—Lee, R. C.-T.: Symbolic Logic and Mechanical Theorem Proving, Academic Press 1973.

[2] Davis, M.—Putnam, H.: A Computing Procedure for Quantification Theory. J. Assoc. Comput. Mach., Vol. 3, 1960, pp. 201–215.

[3] Ezhkova, I.: Is Universal Expert System Possible? Software & Systems. Vol. 1, 1991, No. 2, pp. 19–29.

[4] Fridman, A. Ya.: Situative Approach to Modelling and Structure Control of Nature-Technical Complexes. In: Proceedings 4th International Conference "System Identification and Control Problems" (SICPRO '05), Moscow, Russia (Institute of Control Science, Moscow Russia), 2005, pp. 1075–1108 (in Russian).

[5] Fridman, A. Ya.—Fridman, O. V.: Situative Approach to Modelling of Performance and Safety in Nature-Technical Complexes. In: Juha Lindfors (Ed.): Applied

Information Technology Research – Articles by Cooperative Science Network, University of Lapland, Finland, 2007, pp. 44–59.

[6] Gentzen, G.: Untersuchungen ber das Logische Schließen. Math. Z., Vol. 39, 1934, pp. 176–210, pp. 405–431.

[7] Hilbert, D.—Ackermann, W.: Grundzüge der Theoretischen Logik. Berlin 1928.

[8] Kolmogorov, A. N.—Fovin, S. V.: Elements of the Theory of Functions and Functional Analysis, 1: Metric and Normed Spaces. Graylock, Rochester, N. Y. 1957.

[9] Kulik, B. A.: A Logic Programming System Based on Cortege Algebra. Journal of Computer and Systems Sciences International, Vol. 33, 1995, No. 2, pp. 159–170.

[10] Kulik, B. A.: New Classes of Conjunctive Normal Forms with a Polynomially Recognizable Property of Satisfiability. Automation and Remote Control, Vol. 56, 1995, No. 2, pp. 245–255.

[11] Kulik, B. A.: Representation of Logical Systems in a Probabilistic Space in Terms of Cortege Algebra, 1 – Elements of Cortege algebra. Automation and Remote Control, Vol. 58, 1997, No. 1, pp. 102–114.

[12] Kulik, B. A.: Reliability Analysis of the Systems with Many States on the Basis of the Algebra of Tuples. Automation and Remote Control, Vol. 64, 2003, No. 7, pp. 1029–1034.

[13] Kulik, B. A.: A Generalized Approach to Modelling and Analysis of Intelligent Systems on the Cortege Algebra Basis. In: Proceedings Sixth International Conference on System Identification and Control Problems (SICPRO '07), Moscow, Russia 2007, pp. 679–715 (in Russian).

[14] Kulik, B. A.: N-Tuple Algebra-Based Probabilistic Logic. Systems Analysis and Operations Research, Vol. 46, 2007, No. 1, pp. 111–120.

[15] Kulik, B. A.—Fridman, A.—Zuenko,A.: Logical Analysis of Intelligence Systems by Algebraic Method. In: Cybernetics and Systems 2010: Proceedings of Twentieth European Meeting on Cybernetics and Systems Research (EMCSR 2010), 2010, Vienna, Austria, pp. 198–203 (ISBN 3-85206-178-8).

[16] Mendelson, E.:. Introduction to Mathematical Logic. 4th ed., Chapman & Hall 1997.

[17] Russel, S.—Norvig, P.: Artificial Intelligence: A Modern Approach. Second Edition, Prentice Hall 2003.

[18] Sokolov, B. V.—Fridman, A. Ya.: Integrated Situational Modelling of Industry-Business Processes for Every Stage of Their Life Cycle. In: Proceedings 4th International IEEE Conference "Intelligent Systems" (IS 2008), Varna, Bulgaria 2008, Vol. 1, pp. 8-35–8-40.

[19] Zuenko, A. A.—Fridman, A. Ya.: Logical Inference in Semantic Analysis of Adhoc Path Queries. In: Proceedings 11th National Conference in Artificial Intelligence with Foreign Collaboration (CAI-2008), Dubna, Russia 2008, Vol. 1, pp. 298–304 (in Russian).

[20] Zuenko, A. A.—Fridman, A. Ya.: Context Approach in Support Systems for Open Subject Domains. Artificial Intelligence and Decision Making 3, 2008, pp. 41–51 (in Russian).

[21] Zuenko, A. A.—Fridman, A. Ya.: Development of N-Tuple Algebra for Logical Analysis of Databases with the Use of Two-Place Predicates. Journal of Computer and Systems Sciences International, Vol. 48, 2009, No. 2, pp. 254–261.

Boris Kulik graduated from Leningrad Mining Institute and worked for the USSR Ministry of Geology from 1971 to 1989 in automation of drilling control. Then he took up research on logic, mathematics and artificial intelligence and received his Ph. D. in 1996. Since 1997 he has worked in St. Petersburg Institute of Problems in Machine Science of the Russian Academy of Sciences. He received his Doctor of Science (Physics and Mathematics) degree in 2008. At present, he teaches mathematics at the St. Petersburg University of Culture and Art. He has published 70 scientific papers including 4 monographs.

Alexander Fridman graduated from Leningrad Electro-Technical Institute in 1975 and worked in Baku (Azerbaijan) for Russian Shipbuilding Ministry until 1989, when he moved to Apatity (Murmansk region, Russia) and began working for Russian Academy of Sciences (RAS). He received his Ph. D. in 1976, Doctor of Science degree in 2001 and Professor degree in 2008. At present he is the head of Laboratory on Information Technologies for Control of Industry-Natural Complexes in the Institute for Informatics and Mathematical Modelling of Technological Processes of RAS and Professor of Applied Mathematics Chair in Kola Branch of Petrozavodsk State University. He has 185 scientific publications including 1 monograph, 21 tutorials and 16 certificates for inventions.

Alexander Zuenko a researcher of the Institute for Informatics and Mathematical Modelling of Technological Processes (RAS), graduated from the Petrozavodsk State University in 2005 and received his Ph. D. in 2009. His scientific activities relate to developing software for modelling open subject domains, as well as to knowledge representation and processing. He has 25 scientific publications.