Algebraic Method of Intelligent Data and Knowledge Processing

Boris A. Kulik, Alexander Ya. Fridman, and Alexander A. Zuenko

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.

Index Terms—data processing, knowledge representation, intelligent system, logical inference, multiplace relation.

I. 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 goes 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 the 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.

Below we introduce a mathematical system named n-tuple algebra (NTA) [1], [2] and developed for solving the set of problems described above [3], [4]. We believe that NTA can be used as a base for creating knowledge processing languages. Due to size limits of the article, we expelled all proofs and examples. Those interested can find them in [1]-[4].

II. BASICS OF N-TUPLE ALGEBRA

A. Basic Concepts and Structures

NTA was developed for modelling and analysis of multiplace relations. Unlike relational algebra used for formalization of databases, NTA can use all mathematical logic’s means for logic modelling and analysis of systems, namely logical inference, corollary trueness' check, analysis of hypotheses, abductive inference, etc. NTA 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 that 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. Hereafter attributes are denoted by capital Latin letters which may sometimes have indices, and the values of these attributes are denoted by the 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 $S$ with attributes $X_i$ (i.e. $S = X_1 \times X_2 \times \ldots \times X_n$) the flexible universe will be comprised of different projections i.e. subspaces that use a part of attributes from $S$. 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 in the $T[XYZ]$ 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, $R[XYZ] = [A \times B \times C]$ 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.

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. Another dummy component (D) called an empty set is used in D-n-tuples.

A C-n-tuple that has at least one empty component is empty.

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 because 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. Their formulating in terms of NTA corresponds to the known properties of Cartesian products. Let two homotypic C-n-tuples $P = [P_1, P_2, \ldots, P_n]$ and $Q = [Q_1, Q_2, \ldots, Q_n]$ be given.

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

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

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

(i) $P \subseteq Q$ or $Q \subseteq P$;
(ii) $P_i = Q_i$ for all corresponding pairs of components except one pair.

Note that in NTA, according to Definition 4, equality $P \cup Q = [P_1, P_2, \ldots, P_n]$ 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.

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

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

Definition 5. A complement ($\bar{P}$) of any component $P_i$ 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 $R[XYZ] = [A \times B \times C]$ is given, then $A = X_A, B = Y_B$ and $C = Z_C$.

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

$$\bar{P} = [\bar{P}_1 \ast \ldots \ast]$$

$$[\ast \bar{P}_2 \ast \ldots \ast]$$

$$[\ldots \ast \ldots \ast \bar{P}_n].$$

In the above C-system $\bar{P}$ 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.

We can denote diagonal C-systems as one n-tuple of sets, using reversed square brackets for expressing this. 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.

According to the definition 6, the complement of a C-n-tuple can be directly recorded as a D-n-tuple. For example, let a C-n-tuple $T = \{[b, d] \{f, h\} \{a, b\}]$ be given in the space
\( S = X \times Y \times Z \) where \( X = \{a, b, c, d\}, Y = \{f, g, h\}, Z = \{a, b, c\}.

Then \( T = \{a, c\} \{\text{g} \} \{c\} \}. \) This structure not only allows to compactly denote diagonal \( C\)-systems, but can be also used in some operations and retrieval queries.

**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.

**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.

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. 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 \( p \) corresponding to an attribute \( X_i \) of the initial n-tuple is expressed by a \( D\)-n-tuple (\( C\)-n-tuple) that has having the component \( p \), in the attribute \( X_i \) and dummy components in all the rest attributes.

**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 ones of alternative classes allow to realize all operations of theory of sets on NTA objects without having to represent the objects as sets of elementary n-tuples.

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: renaming of attributes; transposition of attributes and corresponding columns in NTA objects; inversion of NTA objects (for binary relations); addition of a dummy attribute (+Attr); elimination of an attribute (-Attr). Below we introduce these operations and some derivative ones used in logical inference.

**B. 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.

**Inversion of NTA objects.** In case of binary relations, swapping columns without swapping attributes allows to get the relation inverse to the initial one.

**Addition of a dummy attribute** (+Attr) is done when the added attribute is missing in the relation diagram of an NTA object. 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 “\( \emptyset \)” are added into \( C\)-objects, and dummy components “\( \emptyset \)” are added into \( D\)-objects.

**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.

**+Attr and -Attr operations** are used, in particular, for calculating join or composition of two different-type relations defined by NTA objects.

**Join operation** \( R_1[YZ] \oplus R_2[XY] \) 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 \( T \), an n-tuple with relation diagram \( [XYZ] \) is formed from the two n-tuples, the new n-tuple becoming one of the elements of the relational join. Here and below bold letters denote arbitrary sequences of attributes.

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

\[
R_1[YZ] \oplus R_2[XY] = +X(R_1) \cap +Z(R_2).
\]

**Operation of composition** \( R_1[YZ] \circ R_2[XY] \) of relations is performed after calculating their join. For this, we need to eliminate the projection \( Y \) from all elementary n-tuples belonging to the join. In NTA, the composition of relations is calculated according to the formula:

\[
R_1[YZ] \circ R_2[XY] = -(+X(R_1) \cap +Z(R_2)) = -Y(R_1 \circ R_2),
\]

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

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 in this way: \( \cap_G \), \( \cup_G \), \( \emptyset_G \), \( \subseteq_G \), \( \Rightarrow_G \), etc. The first two operations completely correspond to logical operations \( \land \) and \( \lor \). NTA relation \( \subseteq_G \) corresponds to deducibility relation in predicate calculus. Relation \( \Rightarrow_G \) 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.

**III. DATA AND KNOWLEDGE REPRESENTATION IN NTA**

**A. Graphs and Semantic Networks**

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 parallel. Let us now consider the way graphs are expressed in NTA. We will use the graph presented in Fig. 1 as an example.

![Fig. 1. Example of a graph](image)

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 (suppose that it contains \(n\) vertices), which is the graph \(G^\ast\) 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 \(G^\ast\) 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 [4]. 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 Fig. 2) infers an additional link \(R_3\) between the domain of the relation \(R_1\) (vertex \(K\)) and the co-domain of the relation \(R_2\) (vertex \(N\)) as it is shown in Fig. 3 where \(A, B, C\) are variables whose values can be the vertices of the described semantic networks.

![Fig. 2. Initial semantic network](image)

![Fig. 3. Example of a transformation rule for a network](image)

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

B. Correspondence between N-tuple Algebra and Predicate Calculus

In trivial case (when individual attributes do not correspond to multiple relations), an n-tuple corresponds to conjunction of one-place predicates with different variables. For example, a \(C\)-n-tuple \(P_1[XYZ] = \{P_1, P_2, P_3\}\) where \(P_1 \subseteq X\), \(P_2 \subseteq Y\), \(P_3 \subseteq Z\) corresponds to a logical formula \(H = P_1(x) \land P_2(y) \land P_3(z)\). A \(D\)-n-tuple \(\bar{P} = \{ \bar{P}_1, \bar{P}_2, \bar{P}_3 \}\) (corresponds to the negation of the formula \(H\) (disjunction of one-place predicates) \(\neg H = \neg P_1(x) \lor \neg P_2(y) \lor \neg P_3(z)\). 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. An 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. One can find rules of quantification in NTA in [9].

Next section is concerned with logical inference techniques in NTA.

IV. LOGICAL INFERENCE IN NTA

A. Computational Complexity of Algebraic Operations in Logical Inference

A significant number of 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. Identifying cases with polynomially recognizable satisfiability property is of great importance for applied research since it reduces the time required for implementation of algorithms.

The most popular systems of logical inference in mathematical logic are as follows: 1) Hilbert-style calculi proposed in [5]; 2) natural deduction calculus developed by logician G.Gentzen [6]; 3) logical inference based on Resolution Principle that became widely known after the article [7] was published. Logical inference systems often use two theorems introduced and proved in [8]. They are reproduced below since they allow to derive logical corollaries by algebraic methods as well as by inference rules.
Theorem 11. Let formulas $F_1, ..., F_n$ and $G$ be given. Then $G$ is a logical corollary to $F_1, ..., F_n$ if and only if the formula $(F_1 \land ... \land F_n) \supset G$ is a valid one.

Theorem 12. Let formulas $F_1, ..., F_n$ and $G$ be given. Then $G$ is a logical corollary to $F_1, ..., F_n$ if and only if the formula $(F_1 \land ... \land F_n \land \neg G)$ is inconsistent.

Logical inference in NTA is based on the theorems 11 and 12 which can be expressed in NTA terms as follows, since NTA is isomorphic to algebra of sets.

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

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

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

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. We have proved that 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.

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 [2].

The special cases known in mathematical logic can be expressed in NTA structures as well, however, NTA has its own means for reducing laboriousness and computational complexity of algorithms. 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 parallel 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 [2], [4]. 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.

B. 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 $A_1, ..., A_n$ 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:

$$ (A_1 \cap_G ... \cap_G A_n) \subseteq_G B. \tag{1} $$

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 $A = A_1 \cap_G ... \cap_G A_n$, 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 (1). Below we will consider $A$ a C-system. Otherwise it can be transformed into one 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 $B$ should preferably use only a small number of variables from the axioms $A_i, ..., A_n$; 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 the Problem 2.

Decreasing the number of variables in $B_i$ is possible through eliminating some attributes from $A$. Obviously, after this 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. 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 only incomplete projections.

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:
\[ A \rightarrow C, B \rightarrow C, A \lor B. \]

C

It is implied that the formulas below the solidus (corollaries) are derived from the ones above it (axioms). By transforming the conjunction of the formulas above the solidus into a D-system within the \([ABC]\) relation diagram, we get

\[ Up[ABC] = \begin{bmatrix} [0] \otimes [0] \otimes [1] \\ [1] \otimes [0] \otimes [0] \end{bmatrix}. \]

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 \( Up[ABC] \subseteq_G Dn[C] \). Transforming \( Up[ABC] \) into a C-system yields \( Up[ABC] = \begin{bmatrix} [0] \otimes [0] \otimes [1] \\ \ast \otimes [1] \otimes [1] \end{bmatrix} \). 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 \( Up[C] = \begin{bmatrix} [0] \otimes [1] \otimes [1] \\ [1] \otimes [0] \otimes [1] \end{bmatrix} \), 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 \( A \lor B \).

C. Relational Database Management Systems

Let us consider the way DBMS queries are expressed in NTA. Let a DB 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 \( Q_i[Z] = [D] \). We can get the answer to the query by calculating \( P[XYZ] \cap G Q_i[Z] \).

Let us consider an example in which a relation join is required. Suppose that, besides the relation \( P[XYZ] \), our DB contains a relation \( R[XY] \), and we need to find the values \( X \) and \( Y \), if \( Z = a \). In SQL, this is written as \( \text{SELECT } X, Y \text{ FROM } P, R \text{ WHERE } Z = a \text{ AND } P.Y = R.Y \).

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 \( Q_i[Z] = \{ [a] \} \), and the answer to the query are attributes \( X \) and \( Y \), as calculated by this formula:

\[ (P[XYZ] \otimes R[XY]) \cap G Q_i[Z]. \]

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.

V. CONCLUSION

This article suggests using algebraic approach based on general theory of multiplexe 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 modelling 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. Today’s 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 NTA, many declarative commands can be represented as relatively simple procedures. As for implementing logical inference procedures in n-tuple algebra, these 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.

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

- context-oriented database (knowledge base) management systems [10];
- research on additional means of immersing NTA structures into measure spaces [11].

REFERENCES