systems.

2013 | volume 1
issue 3 | 14–38

ISSN 2305-6991
BCSSS

Algebraic Approach to Combining Logical Inference with Defeasible Reasoning

Boris Kulik
Institute of Problems in Mechanical Engineering, Russian Academy of Sciences (RAS), 61 Bol'shoi pr., 199178 St. Petersburg, RUSSIA, ba-kulik@yandex.ru, +7(812)5173498

Alexander Fridman
Institute for Informatics and Mathematical Modelling, Kola Science Centre of RAS, 24A Fersman str., 184209 Apatity Murmansk region, RUSSIA, fridman@iimm.kolasc.net.ru, +7(81555)74050

Alexander Zuenko
Institute for Informatics and Mathematical Modelling, Kola Science Centre of RAS, 24A Fersman str., 184209 Apatity Murmansk region, RUSSIA, zuenko@iimm.kolasc.net.ru, +7(81555)74050

Abstract: The study describes new capabilities of *N*-tuple algebra (NTA) belonging to the class of Boolean algebras and developed by the authors as a theoretical generalization of structures and methods applied in intelligence systems. NTA supports formalization and solving a wide set of logical problems (abductive and modified conclusions, modelling of graphs, semantic networks, expert rules, etc.). Here we mostly focus on unified implementation of logical inference and defeasible reasoning by means of NTA. In NTA, reasoning procedures 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. All NTA reasoning techniques have clear interpretations within classical logic.

Keywords: Data processing; knowledge representation; intelligence system; flexible universe; *n*-ary relation; general theory of relations; n-tuple algebra; logical inference; defeasible reasoning

Acknowledgement: The authors would like to thank the Russian Foundation for Basic Researches (grants 12-07-00302, 12-07-00550, 12-07-00689, 13-07-00318, 14-07-00256, 14-07-00257, 14-07-00205) and the Chair of the Russian Academy of Sciences (project 4.3 "Intelligent Databases" of the Programme # 16 of Basic Scientific Researches) for their aid in partial funding of this research.

At the previous conference (Kulik et al., 2010a), we introduced *n*-tuple algebra (NTA) that uses Cartesian products of sets rather than sets of elements (elementary *n*-tuples) as a basic structure and implements the general theory of *n*-ary relations.

Novelty of our approach is that we developed some new mathematical structures allowing to implement many techniques of semantic and logical analyses; these methods have no analogies in relational algebra and binary relations theory.

A practicable logical analysis should include both deduction (logical inference) and non-deductive analysis techniques, i.e. analyzing uncertainties and inconsistency, as well as forming hypotheses and abductive conclusions. Convenient formal methods of the classical logic provide solution of the deductive tasks only, with some problems arising nonetheless. Other mentioned tasks commonly involve non-classical logics, in particular, the default logic and non-monotonic logics. It is not easy to combine logical inference and non-deductive reasoning within a formal approach. Our general theory of relations provides some possibilities to merge deductive and defeasible analyses.

A.Thayse (1988) defines defeasible reasoning as an opposition to “strongly correct” reasoning. Defeasible reasoning is used when we deal with incomplete, inexact and/or changeable initial information. Non-monotonic and default logics are special cases of formalization for such kinds of reasoning. Conversely, we should also consider methods of modeling and analyzing based on classical logic and suitable for defeasible reasoning. We propose using NTA for this purpose since it unifies solving a wide set of logical problems (Kulik et al., 2010a; 2010b). Below we will focus on performing logical inference and defeasible reasoning by means of NTA. The given paper does not concern inductive methods of logical analysis though they are a part on non-deductive reasoning. We have not completed our research on implementing such methods in NTA yet.

1 Basics of NTA

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 (Codd, 1970; 1972) 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 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 19-25, 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 *n*-ary 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 introduce a mathematical system named *n*-tuple algebra (NTA) (Kulik, 1995a; 1995b) and developed for solving the set of problems described above (Kulik, 2007b; Zuenko & Fridman, 2009). We believe that NTA can be used as a base for creating knowledge processing languages.

1.1 Basic Concepts and Structures

N-tuple algebra was developed for modeling and analysis of *n*-ary 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 *n*-ary 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. 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. $S = X_1 \times X_2 \times \ldots \times X_n$) 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*] = [*A B C*] means that $A \subseteq X, B \subseteq Y, C \subseteq Z$ and *R*[*XYZ*] = *A* × *B* × *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 (∅) 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 calculi, 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 *n*-ary relations as sets rather than element-wise 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 $P=[P_1 P_2 .. P_n]$ and $Q=[Q_1 Q_2 .. Q_n]$ be given.

Theorem 1

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

Examples: [{*b*, *d*} {*f*, *h*} {*a*, *b*}] ∩ [* {*f*, *g*} {*a*, *c*}] = [{*b*, *d*} {*f*} {*a*}];

[{*b*, *d*} {*f*, *h*} {*a*, *b*}] ∩ [* {*g*} {*a*, *c*}] = [{*b*, *d*} ∅ {*a*}] = ∅.

Theorem 2

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

Theorem 3

$P \cup Q \subseteq [P_1 \cup Q_1\ P_2 \cup Q_2 \ldots P_n \cup Q_n]$, 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 $P \cup Q = \begin{bmatrix} P_1 & P_2 & \ldots & P_n \\ Q_1 & Q_2 & \ldots & Q_n \end{bmatrix}$ 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. Let the following two *C*-systems be given in the 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:

[{*a*, *b*, *d*} {*f*, *h*} {*b*}] ∩ [{*a*, *d*} * {*b*, *c*}] = [{*a*, *d*} {*f*, *h*} {*b*}];

[{*a*, *b*, *d*} {*f*, *h*} {*b*}] ∩ [{*b*, *d*} {*f*, *h*} {*a*, *c*}] = ∅;

[{*a*, *b*, *d*} {*f*, *h*} {*b*}] ∩ [{*b*, *c*} {*g*} {*b*}] = ∅;

[{*b*, *c*} * {*a*, *c*}] ∩ [{*a*, *d*} * {*b*, *c*}] = ∅;

[{*b*, *c*} * {*a*, *c*}] ∩ [{*b*, *d*} {*f*, *h*} {*a*, *c*}] =[{*b*} {*f*, *h*} {*a*, *c*}];

[{*b*, *c*} * {*a*, *c*}] ∩ [{*b*, *c*} {*g*} {*b*}] = ∅.

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 (i) or (ii) 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*] = [*A B C*] 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 $P = [P_1 P_2 .. P_n]$

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

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 $\overline{T_1}$ = $]\{a, c\} \varnothing \{c\}[$. In *D*-n-tuples the constant “$\varnothing$” 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 similar 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. End of proof.

For example, the complement of a *C*-system

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

given in a space ***S*** can be calculated as a *D*-system

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

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. 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) that has 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. End of proof.

For example, a *D*-n-tuple $]A \varnothing B C[$ 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 $[A\ B * C]$ – as a *D*-system $\begin{bmatrix} A & \varnothing & \varnothing & \varnothing \\ \varnothing & B & \varnothing & \varnothing \\ \varnothing & \varnothing & \varnothing & C \end{bmatrix}$.

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 ones 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 $P=[p_1 p_2 .. p_n]$ and a *D*-n-tuple $Q=]q_1 q_2 .. q_n[$ 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 [* * ... q_i ... *] 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 an $r_i = p_i \setminus q_i \neq \varnothing$. Consequently, $r_i \subseteq p_i$ and $r_i \subseteq \overline{q}_i$ for every *i*. Then, a non-empty *C*-n-tuple $R=[r_1 r_2 .. r_n]$ exists for which $R \subseteq P$ and $R \subseteq \overline{Q}$, and this proves the impossibility of $P \subseteq Q$. End of proof.

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. End of proof.

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 *n*-ary 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.

1.2 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 operations +Attr and –Attr will be explained below in the section 1.3. These 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 $R_1[\mathbf{V}]$ and $R_2[\mathbf{W}]$ be given, where $\mathbf{V}$ and $\mathbf{W}$ are sets of attributes and $\mathbf{V} \neq \mathbf{W}$. These sets can be separated into nonintersecting subsets with the following transformations:

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

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

Join operation $R_1[\mathbf{YZ}] \oplus R_2[\mathbf{XY}]$ 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 [$\mathbf{Y}$], an *n*-tuple with relation diagram [$\mathbf{XYZ}$] 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

$T_1[\mathbf{YZ}] = (c, d, e, f, g)$; $T_2[\mathbf{XY}] = (a, b, c, d, e)$, and

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

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

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

$$R_1[\mathbf{YZ}] \oplus R_2[\mathbf{XY}] = + \mathbf{X}(R_1) \cap + \mathbf{Z}(R_2). \quad (1)$$

Operation of composition $R_1[\mathbf{YZ}] \circ R_2[\mathbf{XY}]$ of relations is performed after calculating their join. For this, we need to eliminate the projection $[\mathbf{Y}]$ from all elementary *n*-tuples belonging to the join. For example, an elementary *n*-tuple $T_4[\mathbf{XZ}] = (a, b, f, g)$ 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:

$$R_1[\mathbf{YZ}] \circ R_2[\mathbf{XY}] = -\mathbf{Y}(+\mathbf{X}(R_1) \cap +\mathbf{Z}(R_2)) = -\mathbf{Y}(R_1 \oplus R_2), \quad (2)$$

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 $\boldsymbol{S}$:

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

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 $[\mathbf{XZ}]$ 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 in this way: $\cap_G$, $\cup_G$, $\setminus_G$, $\subseteq_G$, $=_G$, 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.

1.3 Correspondence between *N*-tuple Algebra and Predicate Calculus

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

A *D*-n-tuple $\overline{P} =]\overline{P}_1 \overline{P}_2 \overline{P}_3[$ corresponds to the negation of the formula *H* (disjunction of unary 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 $\forall yF(x, z)$ 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 $P[X_1 X_2 \ldots X_n] =]P_1 P_2 \ldots P_n[$ be given. If we add a dummy attribute Y to it, we get $Q[YX_1 X_2 \ldots X_n] =]\varnothing P_1 P_2 \ldots P_n[$. 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 $R[X_1 X_2 \ldots X_n]$ 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 8, the results in projection $[X_1 X_2 \ldots X_n]$ 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, $R_1 = +Y(R) = \forall y(R)$. End of proof.

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

Theorem 14

Let *R*[…*X*…] be a *C*-system that has no *C*-n-tuples with empty components in the *X* attribute. Then for the predicate *P*(…, *x*, …) 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 $R_1, R_2, \ldots R_n$. This means that $R = R_1 \cup R_2 \cup \ldots \cup R_n$. Formula $P = P_1 \vee P_2 \vee \ldots \vee P_n$, where P_i are formulae that correspond to *C*-n-tuples R_i corresponds to this formula in predicate calculus. Applying the $-X$ operation to R, we get $-X(R) = -X(R_1) \cup -X(R_2) \cup \ldots \cup -X(R_n)$.

A formula of predicate calculus $\exists x(P_1) \vee \exists x(P_2) \vee \ldots \vee \exists x(P_n)$ 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 $\exists x(P_1 \vee P_2 \vee \ldots \vee P_n)$, which is ∃x(*P*) after substitution. End of proof.

Theorem 15

Let *R*[…*X*…] be a *D*-system that has no *D*-n-tuples with components "∗" in the *X* attribute. Then for a predicate *P*(…, *x*, …) corresponding to this *D*-system, formula –*X*(*R*) corresponds to the formula ∀*x*(*P*).

Proof

The formula ∀*x*(*P*) is known to be equal to ¬(∃*x*(¬*P*)). A *C*-system $\overline{R}$ that equals to the complement of the *D*-system *R* corresponds to the expression ¬*P*. $Q = -X(\overline{R})$ corresponds to the formula ∃*x*(¬*P*) since $\overline{R}$ satisfies the conditions of Theorem 14. Then ¬(∃*x*(¬*P*)) is an NTA object $\overline{Q}$ 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 $\overline{R}$. End of proof.

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

$$Q[XYZ] = \begin{bmatrix} \{a,b,d\} & \{f,h\} & \{b\} \\ \{b,c\} & * & \{a,c\} \end{bmatrix} \text{ and } \overline{Q}[XYZ] = \begin{bmatrix} \{c\} & \{g\} & \{a,c\} \\ \{a,d\} & \varnothing & \{b\} \end{bmatrix}.$$

$$\text{Then } \exists x(Q[XYZ]) = \begin{bmatrix} \{f,h\} & \{b\} \\ * & \{a,c\} \end{bmatrix} \text{ and } \forall x(\overline{Q}[XYZ]) = \begin{bmatrix} \{g\} & \{a,c\} \\ \varnothing & \{b\} \end{bmatrix}.$$

Let us now analyze some ways of using NTA in reasoning.

2 NTA: Logical Inference Techniques

Logical inference systems often use two theorems introduced and proved in (Chang & Lee, 1973). They are reproduced below since they justify algebraic methods of logical inference.

Theorem 16

Let formulas $F_1,\ldots,F_n$ and *G* be given. Then *G* is a logical corollary of $F_1,\ldots,F_n$ if and only if the formula $((F_1 \wedge \ldots \wedge F_n) \supset G)$ is a valid one.

Theorem 17

Let formulas $F_1,\ldots,F_n$ and *G* be given. Then *G* is a logical corollary of $F_1,\ldots,F_n$ if and only if the formula ($F_1 \wedge \ldots \wedge F_n \wedge \neg G$) is inconsistent.

Logical inference in NTA is based on the theorems 16 and 17. Using correspondences between NTA and predicate calculus, these theorems can be expressed in NTA terms as follows.

Method 1

Let NTA objects $F_1,\ldots,F_n$ and G be given. Then *G* is a logical corollary of $F_1,\ldots,F_n$ if and only if $(F_1 \cap_G \ldots \cap_G F_n) \neq \varnothing$ and $(F_1 \cap_G \ldots \cap_G F_n) \subseteq_G G$.

Method 2

Let NTA objects $F_1,\ldots,F_n$ and G be given. Then G is a logical corollary of $F_1,\ldots,F_n$ if and only if $(F_1 \cap_G \ldots \cap_G F_n) \neq \varnothing$ and $F_1 \cap_G \ldots \cap_G F_n \cap_G \overline{G} = \varnothing$.

Here and below, $\cap_G$, $\cup_G$, $\setminus_G$, $\subseteq_G$, $=_G$, etc. denote generalized operations and relations, i.e. operations and relations with preliminary transformation of attributes in NTA objects. They are similar to corresponding operations and relations of algebra of sets (Kulik et al., 2010a; 2010b).

Thus, deducibility in NTA is based on inclusion or emptiness checks for NTA objects rather than on inference rules. Compared to theorems 16 and 17, methods 1 and 2 contain a precondition $(F_1 \cap_G \ldots \cap_G F_n) \neq \varnothing$, which eliminates situations corresponding to the Duns Scotus' law (the law of denial of the antecedent): falsity implies anything. In specific logical systems, where degeneration of premises causes no problems, this precondition is not obligatory.

Suppose that we have a system of axioms $A_1,\ldots,A_n$ 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:

$$(A_1 \cap_G \ldots \cap_G A_n) \subseteq_G B. \quad (3)$$

This relation allows for correctness checks not only for the inference rules of classical logic, but also for rules specific to a certain knowledge system (Kulik et al., 2010b).

2) Problem of derivation of consequences. In order to solve this problem, we first calculate an NTA object $A = A_1 \cap_G \ldots \cap_G A_n$, then we choose a B_i for which $A \subseteq_G B_i$ is true. We have developed special techniques that allow to find all possible corollaries of a known *A* using the relation (3).

Below we will consider *A* a *C*-system. If it is not true for a given *A*, 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_i 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 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 Bi 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*] = $\begin{bmatrix} \{0\} & \varnothing & \{1\} \\ \varnothing & \{0\} & \{1\} \\ \{1\} & \{1\} & \varnothing \end{bmatrix}$. 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*]. In this case, when the Up is a *D*-system and the inclusion check does not come down to algorithms of polynomial complexity, 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 (Kulik, 1995b).

Transforming Up[*ABC*] into a *C*-system (this operation is often more complex 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 enables us to use algebraic methods for solving problems of logical inference. Moreover, it clarifies 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 (3) 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.

3 Defeasible Reasoning in NTA

Defeasible reasoning belongs to logical systems that admit changes of initial premises during inference. This is due to conjectural nature of some parts of knowledge used in such systems: this knowledge may need improving. The said improvement is not always unambiguous; a suitable hypothesis needs to be chosen from a variety of options. That is why we should define some specific correctness criteria for new knowledge. In classical logic, such criterion is absence of contradictions in the knowledge.

As for mathematical logic, a reasoning system (a theory) is considered inconsistent if and only if both a corollary and its negation follow from the same premises. Conversely, commonsense reasoning, as well as informal scientific one, recognize inference of contrary corollaries (for instance,

if the premises result in both statements "All A are B" and "No A are B") as a definite criterion of inconsistency for a knowledge system. Formally, these statements are not contradictory, since in predicate calculus, negation of the formula $\forall x(A(x) \supset B(x))$ equals $\exists x(A(x) \wedge \neg B(x))$, i.e. "Some A are not B" rather than $\forall x(A(x) \supset \neg B(x))$; the latter corresponds to the statement "No A are B".

3.1 Collisions in Reasoning

In order to eliminate existing discrepancies between formal logic and natural deduction, we propose a concept of collision. Collisions mostly occur during defeasible reasoning when a new knowledge or hypothesis is included into a logical system. They indicate violations of some formal rules or restrictions that control consistency or meaning content of the system. Within the systems with defeasible argumentation, collisions generally correspond to the terms "rebutting", "argument undercutting", "counter-evidence (attack)", etc. The term "collision" was initially used by B. Kulik for analysis of syllogistics-like reasoning, where two kinds of formal collisions are defined, namely:

1. a paradox collision arises if premises infer a statement like "No A are A" ($\overline{A} \subseteq A$), that is, the volume of the term A is empty;
2. a cycle collision occurs when the relation $A \subseteq B \subseteq \ldots \subseteq A$ can be deduced from a system of sets; this means that the terms contained in the cycle are equal.

The listed collisions can be detected without taking the subject domain into account, this is why we named them formal collisions. The third kind of collisions is not a formal one; it features a situation when some consequences do not match some indisputable facts or justified statements. We call this collision an inadequacy collision.

Unlike a logical contradiction which expresses an absolute degeneration of premises, collisions can have opposite interpretations in different cases, i.e. they are semantically dependable. For example, within one system the equality $A = \varnothing$ means an absence of the object that is necessary for existence of the system, and in another system this equality specifies a status of the object A.

The first case requires changing the premises while the second case provides a new useful datum.

Let us adduce examples of collisions, which can happen during analysis of polysyllogisms (Kulik, 2001). We begin with revealing paradox collisions by means of NTA objects.

Example 1

Suppose the following premises are given.

1. All my friends are boasters and not brawlers.
2. All boasters are not self-asserted.
3. All not brawlers are self-asserted.

These premises clearly infer two inconsistent statements, namely 1st and 2nd premises result in the statement (i) "All my friends are not self-asserted", and 1st and 3rd premises infer the statement (ii) "All my friends are self-asserted". If we apply the rule of contraposition to (i), we get the statement "All my friends are not my friends" that shows emptiness of the set of my friends, i.e. reveals a paradox collision.

To analyze collisions in polysyllogistics, one of the authors developed a method based on certain partially ordered sets named *E*-structures (B. Kulik, 2001). However, NTA methods provide solving such tasks as well. Let us express the given premises in predicate calculus. Denote the predicate "*x* is my friend" as *A*, the predicate "*x* is a boaster" as *B*, the predicate "*x* is a brawler" as *C*, and the predicate "*x* is self-asserted" as *D*. Then we can write the premises as follows.

1. $A \supset (B \wedge \overline{C}) = \overline{A} \vee (B \wedge \overline{C})$

2. $B \supset \overline{D} = \overline{B} \vee \overline{D}$.
3. $\overline{C} \supset D = C \vee D$.

These premises correspond to the following NTA objects defined on the universe {0, 1}.

$$P_1[ABC] = \begin{bmatrix} \{0\} & * & * \\ * & \{1\} & \{0\} \end{bmatrix}; \; P_2[BD] = \begin{bmatrix} \{0\} & * \\ * & \{0\} \end{bmatrix}; \; P_3[CD] = \begin{bmatrix} \{1\} & * \\ * & \{1\} \end{bmatrix}.$$

After intersecting these objects we get:

$$P[ABCD] = P_1[ABC] \cap_G P_2[BD] \cap_G P_3[CD] = \begin{bmatrix} \{0\} & * & * & * \\ * & \{1\} & \{0\} & * \end{bmatrix} \cap$$

$$\cap \begin{bmatrix} * & \{0\} & * & * \\ * & * & * & \{0\} \end{bmatrix} \cap \begin{bmatrix} * & * & \{1\} & * \\ * & * & * & \{1\} \end{bmatrix} = \begin{bmatrix} \{0\} & \{0\} & \{1\} & * \\ \{0\} & \{0\} & * & \{1\} \\ \{0\} & * & \{1\} & \{0\} \end{bmatrix}.$$

The first column of the resulting *C*-system contains only the component {0} that indicates a paradox collision $A \supset \overline{A}$. Indeed, the expression $(A \supset \overline{A}) = \overline{A} \vee \overline{A} = \overline{A}$ corresponds to the *C*-n-tuple *S*[*ABCD*] = [{0} * * *] in NTA, and the inclusion $P[ABCD] \subseteq_G \boldsymbol{S}[ABCD]$ is true. That is, the collision $A \supset \overline{A}$ is a consequence from the given system of premises.

If a conclusion contradicts to reality (this is an inadequacy collision), the premises need a correction. In particular, replacing the third premise with "All self-asserted are not brawlers" leads to absence of collision.

Example 2

Now we analyze cycle collisions. Let the premises are as follows.

1. Anything that exists is confirmed experimentally.
2. Anything unknown is not confirmed experimentally.
3. Anything known exists.

Similarly to the previous example, let us use NTA methods to check premises for possible collisions. By *A* denote the predicate "*x* exists", *B* denotes the predicate "*x* is confirmed experimentally", and *C* denotes the predicate "*x* is known". Then the premises will look like

1. $A \supset B = \overline{A} \vee B$.
2. $\overline{C} \supset \overline{B} = C \vee \overline{B}$.
3. $C \supset A = \overline{C} \vee A$.

Rewriting them in NTA terms gives:

$$P_1[AB] = \begin{bmatrix} \{0\} & * \\ * & \{1\} \end{bmatrix}; \; P_2[BC] = \begin{bmatrix} \{0\} & * \\ * & \{1\} \end{bmatrix}; \; P_3[AC] = \begin{bmatrix} \{1\} & * \\ * & \{0\} \end{bmatrix};$$

$$P[ABC] = P_1[AB] \cap_G P_2[CB] \cap_G P_3[AC] = \begin{bmatrix} \{0\} & * & * \\ * & \{1\} & * \end{bmatrix} \cap \begin{bmatrix} * & \{0\} & * \\ * & * & \{1\} \end{bmatrix} \cap$$

$$\begin{bmatrix} \{1\} & * & * \\ * & * & \{0\} \end{bmatrix} = \begin{bmatrix} \{1\} & \{1\} & \{1\} \\ \{0\} & \{0\} & \{0\} \end{bmatrix}.$$

Thus, different attributes take the same truth values in all *n*-tuples which bears witness of equivalence of the statements *A*, *B*, *C*.

In other words, the concluding of consequences showed that the terms "known", "existing" and "confirmed experimentally" were connected in a cycle that detects equivalence of the predicates *A*, *B*,

C. Conversely, this equivalence is not true for the terms "known" and "existing" at least. This is an inadequacy collision requiring a revision of the premises.

Within logical systems exceeding the limits of syllogistics, collisions can model the following cases.

1. Knowledge degeneration: the knowledge turns identically false after inputting new information (in NTA, such a situation shows empty volume of knowledge, and in classical logic it corresponds to the Duns Scotus' law).
2. Degeneration of attributes: inputting new data leads to disappearing of some elements in certain attributes, whereas these elements determine existence of the modeled system. In other words, a check reveals that some meaningful attributes equal to empty set.
3. When new knowledge is input, some different attributes become identical in composition of their elements, which contradicts to the semantics of the system.
4. A discrepancy occurs between the obtained results and some restrictions that are hard to formalize and are described in task settings. For instance, a modeled system can contain some limitations expressed as relations, which must not appear in consequences. If we consider these limitations in initial conditions as complements to the prohibited relations, the whole system can grow significantly more complex. Sometimes, it is simpler to reject prohibited results by comparing them to the prohibited relations.
5. In some cases, collisions can be detected in situations that are normally used for justification of implementing non-monotonic logics. In the standard example about the ostrich Titi (Thayse et al., 1989), two preconditions are given, namely (i) "all birds can fly" and (ii) "Titi is a bird, but it cannot fly". This knowledge does not necessarily require applying a non-classical logic. Moreover, it is possible to detect a collision and correct the premises without breaking laws of classical logic.

It is not easy to foresee all possible kinds of collisions; they can well be unique for some logical systems. We propose the following brief definition for the term "collision".

Collisions are situations occurring during defeasible reasoning when some new knowledge (hypothesis) is inputted. Such situations can be recognized as violations of some formally expressed rules and/or limitations which control consistency and meaning content of the logical system. In particular, it is important in defeasible reasoning systems.

Below, we describe the two most widespread tasks in analysis of defeasible reasoning. They are (i) formation and proof of hypothesis and (ii) search for abductive conclusions.

3.2 Analysis of Hypotheses

NTA allows for a formal definition of hypotheses. Let us suppose that a system of premises expressed as NTA objects $A_1,\ldots,A_n$ is given and the NTA object $A = A_1 \cap_G \ldots \cap_G A_n$ is calculated.

Definition 8

A certain formula *H* is called a hypothesis, if $A_{\cap_G} H$ is false. Here, we assume that the hypothesis is a premise or an axiom.

Otherwise, *H* is a consequence according to (3). Consequently, *H* can be considered as a first approximation hypothesis, if $A \setminus_G H \neq \varnothing$. For the second approximation, we need to check correctness of the hypothesis. The hypothesis is correct if the object $H \cap_G A$ contains no collisions.

Here, we assume that the hypothesis is a premise or an axiom.

Let us consider an example. A book by R. Smullian (1982) contains a set of tasks about a prisoner who had to determine the room where the princess was, and to open this room using certain hints. The problem arose from having a tiger in at least one of the rooms, as the prisoner did not want to meet the tiger. Conversely, meeting the princess would bring the prisoner both freedom and the hand of the princess. The task we analyze below is similar to these tasks from Smullian's book.

Example 3

The prisoner can see three rooms. The tiger is in one of them, the princess is in another one, and the third room is empty. The prisoner has two hints; one of them is false, the second one is true, but nobody knows which hint is true.

Hint #1 tells that the tiger is not in the second room, and the third room is not empty.

Hint #2 says the first room is not empty, and the tiger is not in the second room.

To express hints as *C*-n-tuples, we will use numbers of rooms as attributes with domains {*P*, *T*, *E*}, where values mean:

P – the princess is in the room;

T – the tiger is in the room;

E – the room is empty.

Then the hints denoted by M_1 and M_2 are:

$M_1 = [* \{P, E\} \{P, T\}]; \quad M_2 = [\{P, T\} \{P, E\} *].$

To solve the problem, the two following hypotheses need checking:

Hypothesis # 1 states that M_1 is true and M_2 is false;

Hypothesis # 2 says that M_1 is false and M_2 is true.

Let us consider the first hypothesis. It corresponds to the expression $M_1 \cap \overline{M_2}$. We calculate $\overline{M_2}$ by using suitable NTA theorems.

$$\overline{M_2} =]\{E\}\ \{T\}\ \varnothing[= \begin{bmatrix} \{E\} & * & * \\ * & \{T\} & * \end{bmatrix}$$

Then

$$M_1 \cap \overline{M_2} = [* \{P, E\} \{P, T\}] \cap \begin{bmatrix} \{E\} & * & * \\ * & \{T\} & * \end{bmatrix} = [\{E\} \{P, E\} \{P, T\}].$$

The situation is now a bit simpler. Calculating the Cartesian product $\{E\} \times \{P, E\} \times \{P, T\}$, we see that it contains four elementary *n*-tuples, and only one of them, (*E*, *P*, *T*), meets the conditions of our task. Here we consider occurrence of the same object in different rooms as a collision. Thus, the princess is in the second room.

Now we check the second hypothesis.

$$\overline{M_1} =]\varnothing\ \{T\}\ \{E\}[= \begin{bmatrix} * & \{T\} & * \\ * & * & \{E\} \end{bmatrix}.$$

$$\overline{M_1} \cap M_2 = \begin{bmatrix} * & \{T\} & * \\ * & * & \{E\} \end{bmatrix} \cap [\{P, T\} \{P, E\} *] = [\{P, T\} \{P, E\} \{E\}].$$

Here the *n*-tuple (*T*, *P*, *E*) satisfies the conditions. It differs from the previously found *n*-tuple, but does not change the princess's position. So, both hypotheses results in the same conclusion: the princess is in the second room.

Forming and checking of hypotheses usually accompany other analysis methods for defeasible reasoning. Below, we will describe the use of hypotheses in searching for abductive conclusions.

3.3 Abductive Conclusions

Abduction is a forming of an explanatory hypothesis when we know some of the premises and an estimated consequence that is confirmed with facts or reasonable arguments, but a formal check does not infer it from the given premises. For example, abduction is used during diagnostics.

Discovery of the neutrino is a classical example of abduction. Previously, the scientists had supposed the energy conservation law to be true for experiments in beta decay. However, calculations revealed that the law seemed not to work in this case. In 1930 W. Pauli, a famous physicist, proposed

a hypothesis that some "invisible" particles produced due to beta decay consume some energy. In 1932, E. Fermi named this particle "neutrino". In fact, existence of the neutrino (or to put it more precisely, existence of its antiparticle – the antineutrino) was not experimentally proved until 1953.

Let us now formally define abduction.

Definition 9

If *B* is an estimated consequence of the premises $A_1,\ldots,A_n$ and the statement $A \subseteq_G B$ is known to be false (once again, $A = A_1 \cap_G \ldots \cap_G A_n$), then a formula *H* is an admissible abductive conclusion when the two following conditions are met:

i) *H* is a hypothesis (i.e. $A \subseteq_G H$ is false) and $H \cap_G A$ is not empty;

ii) $(H \cap_G A) \subseteq_G B$, that is, adding *H* into the system of premises results in deducibility of the estimated consequence *B*.

Definition 10

An admissible abductive conclusion is correct if $H \cap_G A$ contains no collisions.

In NTA, it is a practical idea to interpret abduction by means of Venn diagrams. When *B* is not a consequence of *A*, two cases are possible: the intersection of the mentioned sets is empty or non-empty (see Fig. 1). Let us denote the fill area of *A* that equals to $A \setminus_G B$ as *R*.

The right variant in Fig. 1 is a degenerated one. In this case, it is impossible to obtain any admissible abductive conclusion because no hypothesis *H* can satisfy the condition (ii) of Definition 9. For a non-empty intersection of *A* and *B* shown in the left side of Fig. 1, an abductive conclusion is possible to find. In this case, any admissible abductive conclusion does not contain any part of the area *R*, that is $H \subseteq_G \overline{R}$. Otherwise, $H \cap_G A$ cannot be a subset of *B*.

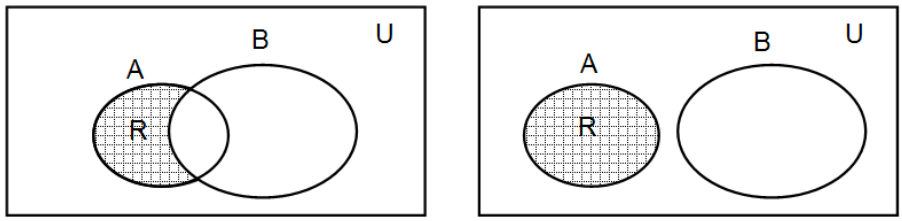

Figure 1: Relationships between premises (A) and a non-deducible consequences (B)

Let R_i be a superset of *R*. R_i is bordered with a dash line in Fig. 2, where two possible cases are shown. Let us choose $H_i = \overline{R_i}$ as a hypothesis because $A \subseteq_G H_i$ is false.

In the first case (left side of Fig. 2), R_i does not completely cover *A*, so the intersection $\overline{R_i} \cap_G A$ is a non-empty set included into *B*, and $\overline{R_i}$ is an admissible abductive conclusion. The correctness of the hypothesis $\overline{R_i}$ can be decided after checking $\overline{R_i} \cap_G A$ for absence of collisions. The degenerated case when $\overline{R_i} \cap_G A = \varnothing$ is shown in Fig. 2 to the right. This case contradicts to Definition 9. Here R_i totally covers both *R* and *A*. Thus, any R_i forming procedure should provide that both $R \subseteq_G R_i$ be true and $A \subseteq_G R_i$ be false.

All of the above results in the following:

Search Algorithm for Abductive Conclusions (Kulik et al., 2010b)

Step 1. Calculate the "remainder" $R = A \setminus_G B$;

Step 2.Build an intermediate object R_i, for which $R \subseteq_G R_i$ is true;

Step 3. Calculate $H_i = \overline{R_i}$ (R_i can now be denoted by $\overline{H_i}$);

Step 4. Calculate $H_i \cap_G A$ and check it for presence of collisions; if they are detected, return to Step 2, otherwise End.

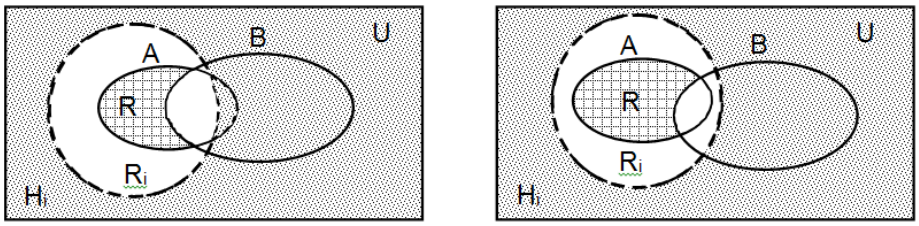

Figure 2: Possible situations in choosing of hypotheses

In Step 2, choice of R_i is not always unambiguous, it depends on situation.

Let us give an example using the rule of dilemma (see above). Suppose that the second premise in the rule of dilemma is missing. We need to retrieve this premise if the first premise, the third one and the "estimated" consequence are given. First, we express the system of known premises by NTA objects: if we denote the given premises $A_1 = A \to C$; $A_2 = A \vee B$ and the "estimated" consequence C, then

$A[X_A X_B X_c] = A_1 \cap A_2 =]\{0\} \varnothing \{1\}[\cap]\{1\} \{1\} \varnothing[= \begin{bmatrix}\{0\} & \varnothing & \{1\}\\ \{1\} & \{1\} & \varnothing\end{bmatrix} = \begin{bmatrix}\{0\} & \{1\} & *\\ \{1\} & * & \{1\}\end{bmatrix}$ and the "estimated" consequence $B[X_A X_B X_c] = [* * \{1\}]$.

Now we have to find a suitable hypothesis (abductive conclusion) solving the problem. Step 1 of the above-introduced search algorithm works as follows.

$R[X_A X_B X_C] = A[X_A X_B X_C] \setminus B[X_A X_B X_C] = \begin{bmatrix}\{0\} & \{1\} & *\\ \{1\} & * & \{1\}\end{bmatrix} \cap \overline{[* \ * \ \{1\}]} = \begin{bmatrix}\{0\} & \{1\} & *\\ \{1\} & * & \{1\}\end{bmatrix} \cap$ $[* * \{0\}] = [\{0\} \{1\} \{0\}]$.

For Step 2, we choose an incomplete projection containing attributes X_B and X_C since the premises include all the rest pairs of attributes.

The hypothesis

$H_1 = \overline{R_1}\ [X_B X_C] = \overline{[\{1\} \ \{0\}]} =]\{0\} \{1\}[= \begin{bmatrix}\{0\} & *\\ * & \{1\}\end{bmatrix}$ corresponds to the chosen projection.

We can write the obtained abductive conclusion as $\overline{B} \vee C$ or $B \to C$. To complete the reasoning, we need to add this conclusion to the initial premises. The result equals the dilemma rule.

4 Discussion at the EMCSR 2012

New capabilities of NTA in unified implementation of logical inference and defeasible reasoning were presented. The TFS vs Algebra formal representations were discussed. It was shown how clarity may be increased by means of logical analysis within frames of classical logic.

The impact of NTA on a more detailed analysis of discrepancies between premises (called “collisions”) was stressed.

It was recommended to the authors that they strive to advance the global positioning of NTA among logical systems, and to developing inductive techniques in NTA.

5 Conclusion

This paper introduces a new mathematical instrument, *n*-tuple algebra, belonging to the class of Boolean algebras. Unlike relational algebra and the theory of binary relations, NTA uses Cartesian product of sets rather than elementary *n*-tuples as a basic structure and implements the general theory of *n*-ary relations.

The novelty of our approach is that we developed some new mathematical structures enabling us to implement many techniques relevant to semantic and logical analyses; these methods have no analogies in conventional theories.

The suggested NTA-based approach substantiates using algebraic methods for solving problems of logical analysis. The algorithm and forming rules for abductive conclusions proposed above can be applied not only to NTA objects expressing formulas of propositional calculus, but also to a more general case when attribute domains contain more than two values. Within a specific knowledge system, choosing variables and their values depends on criteria formed by the content of the system. The techniques that we developed simplify generating abductive conclusions for given limitations, for instance, in composition and number of variables.

References

Chang, Chin-Lang & Lee, Richard Char-Tung. (1973). Symbolic Logic and Mechanical Theorem Proving. New York: Academic Press.

Codd, E.F. (1970). A relational model of data for large shared data banks. Comm. ACM, 13 (6), 377-387.

Codd, E.F. (1972). Relational completeness of data base sublanguages. In Randall Rustin (Ed.), Courant Computer Science Symposium 6: Data Base Systems (pp. 33-64). New York City: Prentice Hill.

Kulik, B.A. (1995). A Logic Programming System Based on Cortege Algebra, Journal of Computer and Systems Sciences International, 33 (2), 1995, pp. 159-170.

Kulik, B.A. (1995). New Classes of Conjunctive Normal Forms with a Polynomially Recognizable Property of Satisfiability, Automation and Remote Control, 56 (2), 1995, pp. 245-255.

Kulik, B. (2001). Logic of Natural Reasoning. Saint Petersburg: Nevsky Dialekt (in Russian).

Kulik, Boris. (2007). 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)

Kulik, B.A. Fridman, A.Ya. Zuenko, A.A. (2010). Logical Analysis of Intelligence Systems by Algebraic Method, in: Proceedings European Meeting on Cybernetics and Systems Research (EMCSR 2010), 2010, Vienna, Austria, pp. 198-204.

Smullyan, Raymond M. (1982). The Lady or the Tiger? And other logic puzzles. New York: Alfred A. Knopf.

Thayse, A., Gribomont, P., Hulin, G. et al. (1988). Approche logique de l'intelligence artificielle, vol.1. De la logique classique a la programmation logique. Paris: Bordas.

Zuenko, A.A. Fridman, A.Ya. (2009). Development of *N*-Tuple Algebra for Logical Analysis of Databases with the Use of Two-Place Predicates, Journal of Computer and Systems Sciences International. 48 (2), 2009, pp. 254-261.

About the Authors

Boris Kulik

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

Alexander Fridman

graduated from the Leningrad Electro-technical Institute in 1975 and worked in Baku (Azerbaijan) for Russian Ship-building Ministry until 1989, when he moved to Apatity (Murmansk region, Russia). He got his PhD in 1976, Doctor of Science degree in 2001 and Professor degree in 2008. At present he is the head of Laboratory on Information Technologies in the Institute for Informatics and Mathematical Modelling of Technological Processes of RAS. He has 230 scientific publications including 4 monographs, 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 got his PhD in 2009. His scientific activities relate to developing software for modelling open subject domains, as well as to knowledge representation and processing. He has 65 scientific publications including 3 monographs, 1 tutorial.