computing@computingonline.net
www.computingonline.net

ISSN 1727-6209
International Journal of Computing

ALGEBRAIC APPROACH TO INFORMATION FUSION IN ONTOLOGY-BASED MODELING SYSTEMS

Irina Artemieva [1], Alexander Zuenko [2], Alexander Fridman [2]

[1] Far Eastern State University, 8 Sushanova str., 690950 Vladivostok, Russia; iartemeva@mail.ru
[2] Institute for Informatics and Mathematical Modelling of Technological Processes of the Russian Academy of Sciences, 24A Fersman str., 184209 Apatity, Russia; {fridman, zuenko}@iimm.kolasc.net.ru

Abstract: *In this paper we discuss the possibilities to use algebraic methods (in particular, n-tuple algebra developed by the authors) to improve the functioning of convenient ontology-based modeling systems. An illustrative example shows the ways to unify representation and processing of two major parts of subject domain ontologies.*

Keywords: *ontology-based system, modeling system, subject domain ontology, n-tuple algebra.*

1. INTRODUCTION

An intelligence system for a domain with complicated structure belongs to knowledge-based systems, which allow for accumulating knowledge relating to different chapters of the domain as well as sub-domains ontologies and data archives in order to support solving various applied tasks by domain specialists.

To represent ontology and knowledge, developers often use graph-oriented structures [1]. Then a graph traversal can help to solve a task since access to information stored in a graph leaf requires finding a path from the root of graph to this leaf. This is time consuming.

Databases (DBs) give an alternative way to represent ontology and knowledge. In this case, we formulate a task solving method by means of a query language. To accelerate access to information, all descriptors of an ontology element including its properties, functions and relations, which form its knowledge base are stored in one database table. To become practically useful for specialists of a complicated subject domain, information components of a program system have to contain data archives, ontology and other domain knowledge. A special data ontology provides interpretation of information stored in archives [2]. Data archives assist in solving different classes of applied tasks including various data analysis for knowledge acquisition.

Using of database tools to represent information components of an intelligence system for subject domains with complicated structure gives a possibility to formulate database queries representing different integrity restrictions on knowledge or/and data as well as rules to coordinate knowledge with data and data with knowledge.

In [3], we considered a method to represent ontology and knowledge by database tables and to use them for developing an intelligence system.

Thus, we can state that developers of modern intelligence systems face certain challenges resulting from fundamentally different approaches used in constructing DBs and knowledge bases (KBs). 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). 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. Algebraic techniques, e.g. those of relational algebra are most commonly used in constructing data processing systems.

Ontology-based systems also have problems with integration their subsystems into a common software environment. In our opinion, algebraic approach seems to be a rational supplement to traditional formal methods in logic in order to unify information representation and processing as well as to improve logical analysis techniques. Below we will mostly dwell on the first part of these problems, namely the unification. As a mathematical and software basis, we use *n-tuple algebra* (NTA) [4, 5] briefly introduced below in Section 5. Then we describe NTA capabilities in dealing with graphs and semantic networks (see Section 6). To exemplify theoretical statements, we refer to chemical subject domain [6] and some training tasks for intelligence systems in this domain.

2. TWO PARTS OF THE DOMAIN ONTOLOGY

Domain ontologies for development of intelligence systems able to not only store, search and edit subject domain ontologies and knowledge but to solve other types of applied tasks as well, have to contain a definition of a notion system used to determine input data for the applied tasks and represent results of solving these tasks. For example, such a notion system for chemistry must allow for describing different properties of physical and chemical processes that take place at a certain period of time and under certain external conditions (see Fig. 1).

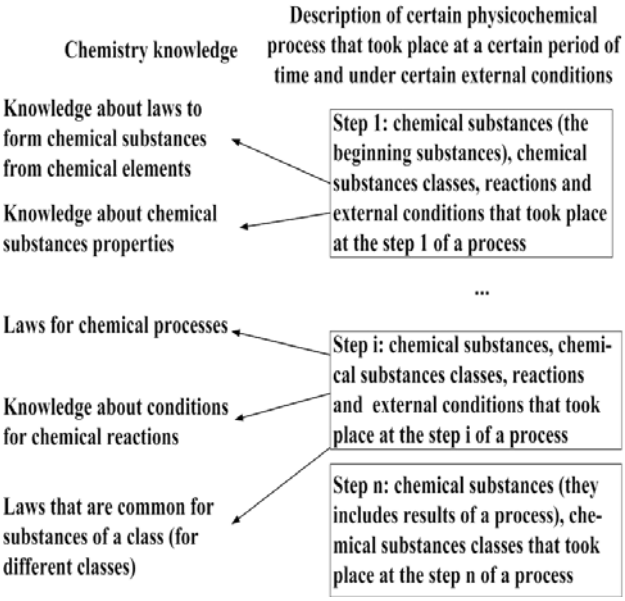

Fig. 1 – Two parts of a chemistry ontology

Thus, the domain ontology can comprise two parts. The first one consists of terms representing the subject domain knowledge. These terms allow us to describe different properties of subject domain objects and to define names of their sets or subsets. For example, terms *Chemical substances, Chemical reactions* are names of nonempty sets; terms *Metal oxides, Metals* are names of subsets. Terms *Atomic weight*, *Current number* specify properties of chemical elements. As another example, let us define the term *Reagents of reaction* as an own property of a chemical reaction. So the definitional domain of the function defining this property is the set of chemical reactions, and the value area is the set of all subsets of all chemical substances. If the term is defined as a property of a reagent of a reaction, then its first argument designates the name of reaction, the second one is the name of a reagent of this reaction.

The second part of the subject domain ontology includes terms, which are used describe different properties of physical and chemical processes that take place at a certain period of time and under certain external conditions. Process descriptions represent results of experiments realized by chemistry researchers. A set of descriptions of such experiments forms data archives, which are information components of intelligence chemistry systems. Data archives allow for testing the regularity of knowledge stored inside an intelligence system during monitoring of information components.

Let us now describe some examples of terms belonging to the second part of the domain ontology. The term *Process reactions* is a function. Its argument is a number of the process step, and the result of the function is a set of reactions that take place at this step. It is obvious that the result of the function is a subset of the set with the name *Chemical reactions.*

Archives can also be used to analyze and generalize experiment results in order to discover new domain knowledge. In this case, data archives either provide input data for a system that automatically realizes such generalizations or take part in regularity tests of the generalizations carried out manually and added to information components of an intelligence system [7].

3. DATABASE STRUCTURE

Information representation structure in a knowledge base is defined by means of the first part of ontology. The subject domain ontology allows to specify the database schema as a set of terms and their interconnections. If a term is defined in the ontology model as a set, it will be represented in a database as a table containing two fields: unique ID (key field) and a value. If the term is defined as a function, it will be expressed as a table where the number of fields is by one greater then the sum of arguments numbers plus the number of elements in the representation of the result (if the result is a Cartesian product rather than a single value, then each element of this product corresponds to one table field). If the result is a predicate, it is regarded as a function with Boolean output.

The type of each field is specified by means of value restrictions from the ontology model [7].

4. TASK TYPES

Here we analyze the constraints, which graph-oriented formalisms of knowledge representation impose on solutions of applied tasks in different subject domains.

In such a case, the constraints, by which the domain ontology may constrain the knowledge contents, have to be expressed as restrictions of

types “part-whole”, “set-subset” and so on. Task solving methods are represented by graph traversal algorithms. Therefore World Wide Web Consortium (W3C) has developed a special formalism to represent other task solving methods as rules [8] for ontology-based program systems.

Let us now consider possibilities we have if we use algebraic methods to represent knowledge.

There are agreements belonging to a set of domain ontological agreements, which can be expressed by equalities. They define relations among values of several terms. Such equalities can be used to automatically manage information components of the intelligence system. They allow to compute values of some terms using given values of all other terms of an equality. This kind of computations can be easily represented by a database query language.

Some typical examples of tasks for finding ways to synthesize substances look as follows. Input data of the tasks can specify:

- a substance to synthesize;
- a set of initial substances (starting points of the synthesis);
- a set of intermediate substances, which can be used during the synthesis.

If a goal substance is specified, the task result can be obtained by a query to the database, which stores information about reactants and results of chemical reactions.

If a set of initial substances is fixed, the task result is a set of reactions whose set of reactants has initial substances as a subset and whose set of reaction results contains the goal substance. If no suitable reactions exist in the database, the task result is a set of reactions’ sequences. The reactants of the first reaction of each sequence contain initial substances as a subset. The results of the last reaction of each sequence contain the goal substance. Each sequence is the result of a query to the database.

A set of intermediate substances forms additional conditions for queries to the database.

Having the structures of the domain ontology and database described, we are going to demonstrate that all necessary structures can be expressed in similar algebraic objects, and the latter ones can be processed by unified algebraic procedures to solve standard tasks of an ontology-based modeling system. However, previously we have to introduce the mathematical basis of these representations and processing procedures.

In the two following sections, we will briefly describe possibilities to use an algebraic system, namely NTA, for solving the problems under discussion.

5. BASICS OF N-TUPLE ALGEBRA

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 the NTA structures as sets of elementary *n*-tuples since every NTA operation uses sets of components of attributes or *n*-tuples of components [9, 10].

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 additional structures providing a compact expression for sets of elementary *n*-tuples.

Names of NTA objects consist of a name proper, sometimes appended with a string of names of attributes in square brackets; these attributes determine the relation diagram in which the *n*-tuple is defined. For instance, if an elementary *n*-tuple $T[XYZ] = (a, b, c)$ is given, then T is the name of the elementary *n*-tuple (a, b, c), X, Y, Z are names of *attributes,* and $[XYZ]$ is the *relation diagram* (i.e. space of attributes), $a \in X$, $b \in Y$ и $c \in Z$. 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 $\boldsymbol{S}$ with attributes X_i (i.e. $\boldsymbol{S} = X_1 \times X_2 \times \ldots \times X_n$), the flexible universe will be comprised of different projections, i.e. subspaces that use a part of attributes from $\boldsymbol{S}$. Every such subspace corresponds to a partial universe.

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

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

A *C-n*-tuple is a set of elementary *n*-tuples; this set can be enumerated by calculating the Cartesian product of the *C-n*-tuple's components. *C-n*-tuples are denoted with square brackets. For example, $R[XYZ] = [A, B, C]$ 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 ($\varnothing$) 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 correspond to the known properties of Cartesian products.

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

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

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

(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 $P \cup Q = \begin{bmatrix} P_1 & P_2 & \dots & P_n \\ Q_1 & Q_2 & \dots & Q_n \end{bmatrix}$ is true for all cases.

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

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

To introduce some algorithms for calculating complements of the NTA objects, we need the following

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.

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

$$\overline{P} = \begin{bmatrix} \overline{P_1} & * & \dots & * \\ * & \overline{P_2} & \dots & * \\ \dots & \dots & \dots & \dots \\ * & * & \dots & \overline{P_n} \end{bmatrix}. \qquad (1)$$

In the above *C*-system (1) 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*. To denote diagonal *C*-systems as one *n*-tuple of sets, we use *reversed* square brackets. 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 Definition 6, the complement of a *C-n*-tuple can be directly recorded as a *D-n*-tuple.

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 some 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 p_i 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.

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 [5, 8, 9, 10].

6. NTA: DATA AND KNOWLEDGE REPRESENTATION

6.1. 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. 2 as an example.

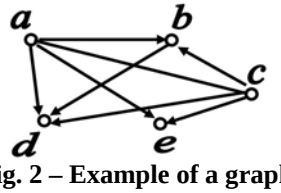

Fig. 2 – Example of a graph

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

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

It is often necessary to determine the set of all the accessible vertices for each vertex of a graph *G*. This information is contained in the *transitive closure* of the graph (suppose that it contains *n* vertices), which is the graph G^+ each of whose vertices is connected with all its accessible vertices with an arc. Transitive closure can be constructed with the following sequence of operations:

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

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

Let us consider the way inference in semantic networks is implemented in NTA [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. 3) 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. 4 where *A*, *B*, *C* are variables whose values can be the vertices of the described semantic networks.

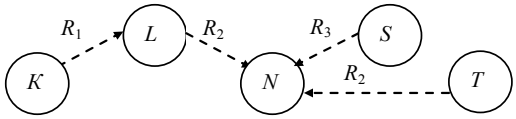

Fig. 3 – Initial semantic network

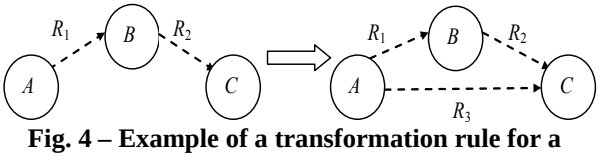

Fig. 4 – Example of a transformation rule for a network

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

6.2. CORRESPONDENCE BETWEEN N-TUPLE ALGEBRA AND PREDICATE CALCULUS

In trivial case (when individual attributes do not correspond to multiplace relations), an *n*-tuple corresponds to *conjunction* of one-place predicates with different variables. For example, a *C*-n-tuple $P[XYZ] = [P_1\ P_2\ P_3]$ 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 one-place predicates) $\neg H = \neg P_1(x) \vee \neg P_2(y) \vee \neg P_3(z)$. An elementary n-tuple that is a part of a non-empty NTA object corresponds to a *satisfying substitution* in a logical formula. An empty NTA object corresponds to an *identically false formula*. An NTA object that equals any particular universe corresponds to a valid formula, or a *tautology*. A non-empty NTA object corresponds to a *satisfiable formula*.

In NTA, attribute domains can be any arbitrary sets that are not necessarily equal to each other. This means that NTA structures correspond to formulas of *many-sorted predicate calculus*. One can find rules of quantification in NTA in [8].

7. VERY SIMPLE CASE STUDY

To clarify the general idea of using NTA in chemistry, we imagined a very simple set of reactions, namely:

$$\begin{aligned} &\text{R1) } MgO + H_2 = Mg + H_2O;\\ &\text{R2) } CuO + H_2 = Cu + H_2O;\\ &\text{R3) } Zn + H_2SO_4 = ZnSO_4 + H_2. \end{aligned} \quad (2)$$

For the given example, these reactions form the set *Chemical reactions* described above in Section 2.

Formalization of this set uses the following attributes and domains corresponding to sorts in predicate calculus and to the term *Chemical substances* from Section 2:

$$\begin{aligned} &\text{S1) Metal oxides } X = \{MgO, CuO\};\\ &\text{S2) Water } Y = \{H_2O\};\\ &\text{S3) Metals } Z = \{Mg, Cu, Zn\};\\ &\text{S4) Salts } W = \{ZnSO_4\};\\ &\text{S5) Gases } V = \{H_2\};\\ &\text{S6) Acids } U = \{H_2SO_4\}. \end{aligned} \quad (3)$$

Let us also define *elementary types* (*elements*) producing substances (3):

$$\begin{aligned} &\text{E1) Oxidizers } P = \{O\};\\ &\text{E2) Acid balances } Q = \{SO_4\}. \end{aligned} \quad (4)$$

The reactions (2) describe two types of relations among *Reagents* and resulting substances with the following diagrams corresponding to the concept of *Chemical reactions* defined as a function in Section 2:

$$\begin{aligned} &\text{F1) Metal reduction } X \times V \rightarrow Z \times Y;\\ &\text{F2) Metal oxidation } Z \times U \rightarrow W \times V. \end{aligned} \quad (5)$$

Considering (4), the diagrams (5) look like

$$\begin{aligned} &\text{F1') Metal reduction } Z \times P \times V \rightarrow Z \times Y;\\ &\text{F2') Metal oxidation } Z \times V \times Q \rightarrow Z \times Q \times V. \end{aligned} \quad (6)$$

These relations constitute the flexible universe [5] of the problem under investigation and form the knowledge base for this example.

Actually, representations (5) and (6) reflect the way to minimize the number of relations in the knowledge base by using variables instead of constants the way that is typical for expert systems. Since our case study is very simple, it requires no variables. So, we will use equations (2), (3) directly.

In NTA terms, reactions (2) look as follows.

$$\text{R1: the left part } R1^{in}[XV] =]\{MgO\}\ \{H_2\}[,$$
$$\text{the right part } R1^{out}[ZY] =]\{Mg\}\ \{H_2O\}[; \quad (7)$$
$$\text{R2: the left part } R2^{in}[XV] =]\{CuO\}\ \{H_2\}[,$$
$$\text{the right part } R2^{out}[ZY] =]\{Cu\}\ \{H_2O\}[; \quad (8)$$
$$\text{R3: the left part } R3^{in}[ZU] =]\{Zn\}\ \{H_2SO_4\}[,$$
$$\text{the right part } R3^{out}[WV] =]\{ZnSO_4\}\ \{H_2\}[. \quad (9)$$

Here we analyze two types of task settings from the set described in Section 4, which are similar to forward and backward inference in expert systems. They are:

T1) Database contains reagents {Zn, H_2SO_4, MgO, CuO}; we need to find all possible resulting reactions and substances;

T2) It is necessary to find out whether {Mg} can be produced from the given reagents {MgO, Zn, H_2SO_4}.

T1 implementation

Initial data stored in the DB can be recorded as a *D-n*-tuple because they can be used independently:

$$DBase[XZU] =]\{MgO, CuO\}\{Zn\}\{H_2SO_4\}[. \quad (10)$$

To apply each of the rules (2), we need to pass through two stages. First, the solver compares the left part of a rule Ri^{in} with the current DB contents

Dbase. If they match, the DB is modified according to the right part of the rule Ri^{out}. Let us assume that we just add substances from the right part into the DB.

To check truthfulness of the left side of a rule in NTA, we have to find out whether the relation

$$Ri^{in} \subseteq_G Dbase \quad (11)$$

is true. If it is true, the solver corrects the DB this way:

$$Dbase := Dbase \cup_G Ri^{out}. \quad (12)$$

If the solver checks the rules (2) similarly to Markov's algorithm (starting from the first rule and applying the first applicable one), it will decide that the first and the second rule are not applicable as the check of the relation (11) for them gives the negative result. For instance, for the first rule this check will look like

$$]\{MgO\}\ \{H_2\} \varnothing \varnothing[\subseteq]\{MgO, CuO\} \varnothing\ \{Zn\}\ (H_2SO_4\}[\quad (13)$$

where relations are reduced to the same diagram [*XVZU*].

Checking the third rule within the diagram [*XZU*]:

$$]\varnothing\ Zn\ H_2SO_4[\subseteq]\{MgO, CuO\}\ Zn\ H_2SO_4[\quad (14)$$

will give the positive answer, and the DB will be modified according to (12):

$$DBase[XZUWV] = [DBase[XZU] \cup_G R3^{out}[WV] = \\ =]\{MgO, CuO\}\ \{Zn\}\ \{H_2SO_4\}\ \{ZnSO_4\}\ \{H_2\}[. \quad (15)$$

T2 implementation

In NTA, this task is realized by recursive passing through the following steps.

1. Setting the initial data.
2. Setting the goal data.
3. Inclusion check of the goal data in right parts of rules. If the result is negative, backtrack to the closest previous fork of the derivation tree. If there is no possible backtracking, the inference is over with the negative answer.
4. If the inclusion check finds a suitable rule, its left part has to replace its right part in the set of goals. When the set of goals contains initial (leaf) data only, its inclusion into the DB is checked. The positive (negative) result of the inclusion check witnesses the positive (negative) reply to the query.

For our example, the initial data can be expressed as a *D*-*n*-tuple

$$S =]\{MgO\}\ \{Zn\}\ \{H_2SO_4\}[, \quad (16)$$

and the initial set of goals (at the step number 0) equals

$$G^0 =]\{Mg\}[. \quad (17)$$

Comparing (17) with the right parts of the rules (2), the solver finds the only suitable rule (7), for which

$$G^0 \subseteq_G R1^{out}[ZY] \quad (18)$$

is true and forms the next goal set as

$$G^1 =]\{MgO\}\ \{H_2\}[. \quad (19)$$

Then the solver finds $\{H_2\}$ in $R3^{out}[WV]$ (9) and forms

$$G^2 =]\{MgO\}\ \{Zn\}\ \{H_2SO_4\}\ [. \quad (20)$$

The set (20) contains leaf data only (no its components belong to right parts of the rules (2)), and $G^2 \subseteq_G S$ is true. So, the inference answers "yes" to the query.

For tasks T1 and T2, consequences of admissible reactions can be formed by saving numbers of applied rules.

8. CONCLUSION

NTA provides storing and processing of both data and knowledge structures by similar techniques. The novelty of NTA lies in creating some new mathematical structures allowing to represent both data and knowledge. This feature simplifies combining data and knowledge bases within a single software system. In NTA, the subject domain ontology and various factual information can be recorded as a number of multiplace relations, and different queries are expressed by some operations analogous to those of theory of sets. The main idea of this processing is as follows. An initial relation defined on a Cartesian product *D* can be often split into blocks corresponding to relations on some projections of *D*, which greatly reduces laboriousness of operations on this relation by using its matrix properties. This allows to process every block separately using known features of Cartesian products, for instance, by paralleling the necessary operations. This idea provides new opportunities to express complex methods of reasoning by comparatively simple algorithms, which can be easily modelled in the computer.

If all information is stored in NTA objects of the same class (like *D*-*n*-tuples used for the case study in

Section 6), these operations are polynomially complex [4]. Generally speaking, traversals of a derivation tree in NTA results in NP-complete algorithms just as in conventional approaches. Besides usage of matrix properties of NTA objects, new structural and statistical classes of conjunctive normal forms with polynomially identifiable satisfiability properties were discovered in NTA. Consequently, we can implement many algorithms whose complexity evaluation is theoretically high, e.g. exponential, in polynomial time, on the average. As for making databases more intelligent, NTA can be considered an extension of relational algebra to knowledge processing. In the authors' opinion, NTA can become a methodological basis for creating knowledge processing languages.

NTA structures provide expressing multiplace predicates but have ordinary sets as components, these sets corresponding to unary predicates. Algebra of conditional *n*-tuples [9] was developed to expand abilities of NTA by dealing with binary predicates.

In this paper, we have illustrated solving of comparatively simple tasks in chemical synthesis when only initial and goal substances are fixed. Solving some harder tasks where we must use certain intermediate products during the synthesis, will probably require for abductive-like reasoning [5], but this matter needs additional studies we plan to conduct in near future.

9. ACKNOWLEDGEMENTS

This work was supported in part by the Russian Foundation for Basic Researches (grant 11-08-00641), OITVS of the Russian Academy of Sciences (RAS) (project # 2.3) and the Chair of RAS (project 4.3 of the Programme # 15).

10. REFERENCES

[1] M. Denny, Ontology Building: a Survey of Editing Tools: http: //www.xml.com/pub/a/2004/07/14/onto.html.

[2] A. Kleshchev, I. Artemjeva, Mathematical models of domain ontologies, *Int. Journal on Inf. Theories and Appl.* (14) 1 (2007), pp. 35-43.

[3] I. Artemieva, N. Reshtanenko, V. Tsvenikov, Upgradable tree levels editor of metaontologies, ontologies and knowledge for chemistry, *Int. Book Series "Information Science and Computing"* (5) 2 (2008), pp. 119-124.

[4] B. Kulik, A. Fridman, A. Zuenko, Logical Analysis of Intelligence Systems by Algebraic Method, *Proceedings of Twentieth European Meeting on Cybernetics and Systems Research (EMCSR 2010)*, Vienna, Austria 6-9 April 2010, pp. 198-203.

[5] B. Kulik, A. Zuenko, A. Fridman, Modified reasoning by means of N-tuple algebra, *Proceedings of the 11th International Conference "Pattern Recognition and Information Processing (PRIP'2011)",* Minsk, Republic of Belarus 18- 20 May 2011, pp. 271-274.

[6] I. Artemieva, Multilevel modular chemistry ontology: structure and management, *Proceedings of the First Russia and Pacific Conf. on Computer Technology and Applications (RPC 2010)*, Vladivostok, Russia 6-9 September 2010, pp. 12-17.

[7] I. Artemieva, A. Zuenko, A. Fridman, Integration of ontologies, knowledge and data archives into ontology-based modeling systems, *Proceedings of the 11th International Conference "Pattern Recognition and Information Processing (PRIP'2011)",* Minsk, Republic of Belarus 18- 20 May 2011, pp. 303-306.

[8] *RIF Production Rule Dialect.* W3C Working Draft 11 February 2010. URL: http://www.w3.org/TR/2010/WD-rif-prd-20100211/.

[9] B. Kulik, A generalized approach to modelling and analysis of intelligent systems on the cortege algebra basis, *Proceedings Sixth International Conference on System Identification and Control Problems (SICPRO'07)*, Moscow, Russia, 2007, pp. 679-715. (in Russian).

[10] A. Zuenko, A. Fridman, 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.

Irina L. Artemieva *graduated from Far Eastern State University and worked for the Institute of Automation and Control Processes of the Far Eastern Branch of the Russian Academy of Sciences from 1978 to 2010. Since 2010, she works for the Institute of Applied Mathematics of the FEBRAS. Her scientific interests are within artificial intelligence. She got her PhD in 1992, her Doctor of Science (Computer Science) degree in 2009 and a Professor degree in 2011. At present she is a Professor of Far Eastern Federal University (FEFU). She has published 180 scientific papers.*

Alexander Zuenko, *a researcher of the Institute for Informatics and Mathematical Modelling of Technological Processes (IIMM) of the Russian Academy of Sciences (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 35 scientific publications including 2 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) and began working for RAS. He got his PhD in 1976, a Doctor of Science (Technical Sciences) degree in 2001 and a Professor degree in 2008. At present he is the head of Laboratory on Information Technologies for Control of Industry-Natural Complexes in the Institute for Informatics and Mathematical Modelling of Technological Processes of RAS and professor of Applied Mathematics Chair in Kola Branch of the Petrozavodsk State University. His scientific interests include modelling and intelligence systems. He has 215 scientific publications including 3 monographs, 21 tutorials and 16 certificates for inventions.*