International Journal of Engineering and Advanced Technology (IJEAT)
ISSN: 2249-8958 (Online), Volume-9 Issue-2, December, 2019

Functions in Column-Based Intelligent Systems

A.M. Chesnokov

Abstract: ***The possibility of implementing functions in column-based intelligent systems have been considered in the article. The basic concepts and definitions have been provided. The representation of functions in such systems have been proposed and the problem of implementing functions has been formulated. The solution to the problem of the implementation of functions using the element-by-element comparison method and the intersection method has been provided, as well as an assessment of the complexity of the solution.***

Keywords: ***artificial intelligence, column-based intelligent systems, column, function.***

I. INTRODUCTION

Column-based intelligent systems are an example of systems whose primary function is pattern memorizing. Such systems are not widespread since at first glance it seems that such systems, in addition to memorizing patterns, cannot solve any other problems. Column-based intelligent systems prove that this is not the case. This paper shows the possibility of implementing complex functional dependencies in such systems. Column-based intelligent systems are systems considered in the framework of the following model [3, 5]. There is, although very large, yet finite *name set* U intended for naming objects of arbitrary nature. Without loss of generality, it is believed that a set of names U is a subset of a set of integers Z. In the set of names U, disjoint subsets that are called *name domains* are distinguished. The number of allocated name domains is not constant. New name domains can be entered at any time, and additional elements can be added to any name domain. The allocation of name domains in real subject areas can be caused by various reasons (e.g., typification). One of the most important reasons is the need to ensure the absence of random coincidence of names in various parts of a large-scale system. Any finite set of names belonging to one or another name domain is called a *pattern*.

The patterns of any pattern set P can be renumbered using the names of a certain name domain U' for this:

$$P = \{p_i \mid i \in U'\}.$$

An ordered pair (i, p_i) is called a *column*. A column is designated as $(i \mid p_i)$, where i – column name, p_i – column pattern. Also, the designation $i \to p_i$ is used. In this case, it is said that the column name i is a *reference* or a *pointer* to column pattern p_i. In turn, about the pattern p_i itself, it will be said that this pattern has a name i or is known by name i.

Revised Manuscript Received on December 30, 2019.

* Correspondence Author
A.M. Chesnokov*, Institute of Control Science of RAS , Moscow, Cand, Russia.

The mapping $\varphi : i \to p_i$ is called the *name mapping*. By default, it is considered that the name mapping is one-to-one. All cases when this is not so, are specified.
To name an pattern p using a name i, or to assign a name i to an pattern p, means that such an addition is made to the definition of the name mapping φ that $\varphi(i) = p$.

A name i that has not yet been used to name patterns is called a *pure* or *empty* name. It can be represented as a column having an empty pattern, i.e. a column of the type $(i \mid \varnothing)$ or $i \to \varnothing$, where $\varnothing$ is an empty set.
Column patterns include the names of other columns as well as pure names. Therefore, the column pattern consists entirely of the names of other columns, each of which serves as a pointer to the corresponding pattern, possibly empty. In turn, any name from a non-empty pattern also points to its pattern, etc. The result is a complex column structure.
An *index* is any finite set of columns. The composition of any index can be changed by adding or removing columns. These operations are called index *addition* and *deduction* and are denoted by + and −. In the following example l columns $(i_k \mid p_k)$ are added to the index A:

$$A + \sum_{k=1}^{l} (i_k \mid p_k).$$

Obviously, the index can be represented in the form of a table consisting of entries of the form “column name − names included in the column pattern”. Such a table in vertical form consists of columns of variable height. In the bottom row of the table, under the fraction bar, there are column names. Above the name of each column, all the names included in the pattern of the column are listed. By default, column names and names in patterns are considered to belong to different name domains.
If the patterns are unordered name sets, then the order in which the names are written in the column patterns can be arbitrary. If the patterns are ordered, then the names in the column patterns are written in a certain order, for example, from bottom to top, i.e. the first pattern name in the first row above the fraction bar, the second in the second, etc. Below, as a simple example, an index A is provided, consisting of three columns $(1 \mid \{1, 3\})$, $(1 \mid \{2, 3, 4\})$ and $(3 \mid \{4, 5\})$ with patterns in the form of unordered sets.

$$\begin{array}{ccc} & A & \\ & 4 & \\ 3 & 3 & 5 \\ 1 & 2 & 4 \\ \hline 1 & 2 & 3 \end{array}$$

A column-based intelligent system is one or more indexes and a mechanism that works with them called a *column engine*. Receiving information about the outside world in the form of patterns, the column engine forms new columns, changes existing ones, removes unnecessary ones and performs all other column operations.

The knowledge in the systems under consideration is represented using columns, and the process of knowledge accumulation is based on memorization of new patterns under certain names. In this case, *the elementary basic problems*, without the solution of which the functioning of the system is impossible, are obviously the *direct problem* (given a pattern, obtain its name), and the *inverse problem* (given a name, obtain the corresponding pattern). Memorization of new patterns is performed as part of a direct problem. If an unnamed pattern is found during its solution, the column engine assigns a certain name to it and saves the corresponding data.

From a formal point of view, memorization of any pattern under a certain name always means forming the corresponding column $(i \mid p_i)$. At the same time, this does not mean that the data will be stored in this form inside the system. The internal representation of the stored data is determined only by the method of solving basic problems, the method of its implementation and may differ significantly from the formal description in the form of a column $(i \mid p_i)$. An example of a method for which the formal description coincides with the internal data representation is a method based on element-by-element comparison of patterns [3–5]. In other cases, the formed column $(i \mid p_i)$, most likely will be stored in an implicit form, and the solution of basic problems will not only show its existence, but will also allow you to get its pattern by the column name, and its name by the pattern.

Solving the basic problems, the column engine actually follows the links $p_i \to i$ in the direct problem, and $i \to p_i$ in the inverse problem. This provides the basis on which the solution to all other problems is based. Solving any kind of such problem basically presents a chain of link following until the result is obtained.

Since everything is finite in the model under consideration, the solution of basic problems always exists. Moreover, the general universal method for solving them is the already mentioned method based on the element-by-element comparison of patterns. From the point of view of theory, this is enough to assess the possibilities of solving various problems with the help of column-based intelligent systems. However, if we talk about the practical application of such systems, more effective methods for solving basic problems, especially high dimension problems, are needed.

One of the possible methods to more effectively solve basic problems is the *intersection method*. The idea of the intersection method goes back to the book index. In it, for each heading, there are many pointers to those pages of the book where this heading can be found. The query from several headings obviously corresponds to the intersection of the sets of pointers for these headings.

In the early 2000s A.M. Mikhailov showed that the intersection method can be used to work with patterns [7, 8]. Within the framework of the emerging direction, called the index approach, the intersection method is mainly used in solving recognition problems [1, 2, 9].

Based on the results of [7, 8], the author proposed a variant of the intersection method for research in the field of column-based intelligent systems [3, 5]. For this option, necessary and sufficient conditions for the existence of a solution to a direct problem were obtained, the implementation of which has little effect on the universality of the method. This variant of the intersection method is also characterized by the complete absence of the need for element-by-element comparison of patterns.

It should be emphasized that the intersection method is just one of the possible methods for solving basic problems. Instead, other methods and tools can be used, in particular, hardware-software, providing high efficiency in solving basic problems of this or that type.

In the [3–6] works, the solution of various basic problems was considered for patterns in the form of unordered finite sets, for patterns in the form of vectors or finite sequences, and also for patterns in the form of finite multisets. The ability of the column-based intelligent systems to work under incomplete information was shown [4]. It turned out that the prediction is an innate property of such systems. In [3, 5] it was proved that there is the possibility of implementing Boolean functions $f: B^n \to B$, where $B = \{0, 1\}$. This work is devoted to the possibility of implementing arbitrary functions $f: U^n \to U^m$, or, more precisely, arbitrary functions of type $f: U_1 \times \ldots \times U_n \to U_{v1} \times \ldots \times U_{vm}$, where U_j – arguments' name domains, $j = 1, \ldots, n$, U_{vj_v} – values' name domains, $j_v = 1, \ldots, m$. The following section provides basic definitions and considers the representation of functions in column-based intelligent systems. Then, the implementation problem is formulated for the functions and its solution is given using the method of element-by-element comparison of patterns and the intersection method.

II. REPRESENTATION OF FUNCTIONS IN COLUMN-BASED INTELLIGENT SYSTEMS

Function

$$f: U_1 \times \ldots \times U_n \to U_{v1} \times \ldots \times U_{vm}$$

called *n-place*, or *n-ary*, and is defined as a finite set of ordered pairs (p, p_v), where $p \in U_1 \times \ldots \times U_n$, U_j – arguments' name domains, $j = 1, \ldots, n$, $p_v \in U_{v1} \times \ldots \times U_{vm}$, U_{vj_v} – values' name domains, $j_v = 1, \ldots, m$. A set consisting of all patterns p in pairs (p, p_v) of function f, is called the *domain* of this function and is denoted by X_f. Similarly, a set consisting of all patterns p_v in pairs (p, p_v) of function f, is called the *codomain* of this function and is denoted by Y_f. The set of *n*-ary functions with the dimension m of codomain patterns is denoted $\mathcal{F}(n, m)$.

Let some function be given $f \in \mathcal{F}(n,m)$. All patterns $p \in X_f$ can be named, using the names $i_p \in U_p$, where U_p is the name domain for patterns $p \in U_1 \times \ldots \times U_n$. Substituting the patterns $p \in X_f$ with their names i_p, we obtain the definition of the function f in the form

$$f = \{(i_{pk}, p_{vk}) \mid k = 1, \ldots, l\},$$

where $l = |X_f|$, $|\cdot|$ – the number of elements (potency) of the set. Moreover, both available definitions of the function are equivalent, since there is a one-to-one correspondence between them. One-to-one mapping ϕ_f is defined as:

$$\phi_f(\{(i_{p1}, p_{v1}), \ldots, (i_{pl}, p_{vl})\}) = \{(\varphi_p(i_{p1}), p_{v1}), \ldots, (\varphi_p(i_{pl}), p_{vl})\}$$
,

where $\varphi_p : i_p \to p$ – name mapping for patterns p.

The purpose of further transformations is a sequence of equivalent function definitions that will provide a solution to the implementation problem.

Both available function definitions are not suitable for this, as they are abstract. They do not contain *an explicit pointer* to that particular function for which the function value will be calculated. Such a pointer, playing the role of an integer identifier, in our case, will be the name of the function.

Bringing function f to the form with an explicit indication of its name, we obtain:

$$f : i_{pk} \to p_{vk} \Rightarrow (i_F, i_{pk}) \to p_{vk},$$

where $i_F \in U_F$ – name of function f, U_F – functions' name domain. Definition of the function f is transformed into a finite set of ordered pairs of the form:

$$f = \{((i_F, i_{pk}), p_{vk}) \mid k = 1, \ldots, l\}.$$

This definition is the starting point for the problem of implementing functions.

This definition can be simplified by naming two-dimensional patterns (i_F, i_{pk}) using the names of some name domain U_d. Definition of the function f will look like:

$$f = \{(i_{dk}, p_{vk}) \mid k = 1, \ldots, l\},$$

where $i_{dk} \in U_d$ – name of the pattern (i_F, i_{pk}). In this case, ordered pairs (i_{dk}, p_{vk}) can be considered as columns of the form $(i_{dk} \mid p_{vk})$. Therefore, the last definition of a function is an index

$$F = \{(i_{dk} \mid p_{vk}) \mid k = 1, \ldots, l\},$$

which one-to-one corresponds to the original representation of the function by name i_F. One-to-one mapping $\phi_F : F \to f$ is defined as:

$$\phi_F(F) = \{(\varphi_d(i_{d1}), p_{v1}), \ldots, (\varphi_d(i_{dl}), p_{vl})\},$$

where $\varphi_d : i_d \to (i_F, i_p)$ – name mapping for patterns (i_F, i_p).

The resulting representation of the function as an index F provides a solution to the problem of implementing arbitrary functions $f \in \mathcal{F}(n,m)$.

III. PROBLEM OF THE IMPLEMENTATION OF THE FUNCTIONS

Let's say that $\mathcal{F}$ is a set of functions $f \in \mathcal{F}(m, m_v)$, $1 \le m \le n, 1 \le m_v \le n$. Let's define through P so many patterns that for any function $f \in \mathcal{F}$ domain is $X_f \subset P$.

The problem of implementing functions is formulated as follows.

Let there be a certain number of functions $f \in \mathcal{F}$, which the system remembered under certain names from the name domain U_F. The set of names of these functions is denoted by U'_F, $U'_F \subset U_F$. When solving the implementation problem for any function by name $i_F \in U'_F$ it is essential for any pattern $p \in P$ to define if pattern p belongs to domain X_f of the function by the name i_F. If pattern is $p \in X_f$, it is necessary to find the value of this function for the pattern p.

The sequence of equivalent representations of the function obtained above determines both the scheme for memorizing a new function and the scheme for solving the implementation problem.

Let us assume that under the name $i_F \in U_F$ a new function $f \in \mathcal{F}$ must be memorized. First, for all patterns of the domain of this function X_f a direct problem is solved and their names i_p are found. After that $l = |X_f|$ of the patterns (i_F, i_p), are formed for which also a direct problem is solved and their names i_d are found. Eventually, into the index A_F l of the columns $(i_d \mid p_v)$ is added, where $A_F = \sum F_j$ – union of indices F for different functions $f \in \mathcal{F}$, p_v – value of the function by the name i_F for the pattern by the name i_p, i_d – name of the pair (i_F, i_p). Now let us assume that for some function under the name $i_F \in U'_F$ an implementation problem must be solved for $\forall p \in P$. First for the pattern p a direct problem is solved and its name i_p is found. Then the pattern (i_F, i_p) is formed for which the direct problem is solved and its name i_d is found.

A column $(i_d \mid a_{vd})$ of the index $A_F = \sum F_j$ is taken. The pattern a_{vd} of this column is the desired value of function by name i_F for the pattern p. If in the process of solving one of the two specified direct problems a nameless new pattern is found, then the value of the function cannot be found, since, obviously, the pattern p does not belong to the domain of the function by name i_F.

IV. SOLVING THE PROBLEM OF IMPLEMENTING FUNCTIONS USING THE ELEMENT-BY-ELEMENT COMPARISON METHOD

When solving the implementation problem using the element-by-element comparison method, the system will use indices A, A_d and A_F. Index A will be used to store the patterns p_k, index A_d to store patterns (i_F, i_{pk}), and index $A_F = \sum F_j$ will consist of the columns $(i_{dk} \mid p_{vk})$ for different functions $f \in \mathcal{F}$.

In the initial state $A = \varnothing$, $A_d = \varnothing$ and $A_F = \varnothing$.

Assume $i_F \in U_F$ is a pure name, by which a new function $f = \{(p_k, p_{vk}) \mid k = 1, ..., l\}$ must be memorized, where $l = |X_f|$. First for all the patterns $p_k \in X_f$ a direct problem is solved. Every pattern p_k is element-by-element compared with the patterns of all the index A columns. If the match is found, then the name i_p of the column $(i_p \mid a_p) \in A$ is such, that $p_k = a_p$, is the name of the pattern p_k. In case the match is not found, pattern p_k is a new one and it must be memorized. Columns engine chooses any pure name $i_{pk} \in U_p$ for it and carries out addition $A + (i_{pk} \mid p_k)$, that is, to the index A a column $(i_{pk} \mid p_k)$ is added. A solution of the direct problem is name i_{pk}, by which pattern p_k will be known.

After that a direct problem is solved for the patterns (i_F, i_{pk}). Since f is a new function, all these patterns are new and they must be memorized. For each of them the columns engine chooses pure name $i_{dk} \in U_d$ and carries out additions

$$A_d + \sum_{k=1}^{l} (i_{dk} \mid (i_F, i_{pk})),$$

$$A_F + \sum_{k=1}^{l} (i_{dk} \mid p_{vk}).$$

That is, to index A_d l of the columns $(i_{dk} \mid (i_F, i_{pk}))$ is added, while to index A_F l of the columns $(i_{dk} \mid p_{vk})$ is added, where p_{vk} is a value of the function f for pair (i_F, i_{pk}), known by the name i_{dk}.

Similarly, all other new functions $f \in \mathcal{F}$ are memorized.

Let us assume it be necessary to solve the implementation problem and find the value of the function by name $i_F \in U'_F$ for any pattern $p \in P$, where $U'_F \subset U_F$ is a set of names of all the functions known to the system. For pattern p a direct problem is solved. It is element-by-element compared with the patterns of all the columns of the index A. If the matches are not found, then the pattern p is not included in the domain of any of the known functions. Otherwise the name of the column $(i_p \mid a_p) \in A$ is such, that $p = a_p$, is a name of the pattern p. After that the pattern (i_F, i_p) is formed, for which a direct problem is also solved. It is element-by-element compared with the patterns of all the columns $(i_d \mid a_d) \in A_d$. If the matches are not found, then the pattern p is not included in the domain of the function by the name i_F. Otherwise the column name i_d, for which the match is found, is the name of pattern (i_F, i_p). After that the column $(i_d \mid a_{vd}) \in A_F$ is taken. Pattern of this column a_{vd} is a value of the function by the name i_F for pattern p.

V. SOLVING THE PROBLEM OF IMPLEMENTING FUNCTIONS USING THE INTERSECTION METHOD

The general scheme for memorizing functions and solving the implementation problem remains unchanged. It includes solving two direct problems – for patterns $p_k \in P$ and for patterns (i_F, i_{pk}), where i_{pk} – name of the pattern p_k. In order to solve the direct problem for patterns p_k using the intersection method, index $A = \{A_1, ..., A_n\}$ will be used, as well as function $m(i_p)$ given by a set of ordered pairs (i_p, m), which will store the dimensions of known patterns p_k [3–5]. Besides, index $A_d = \{A_{d1}, A_{d2}\}$ will be used in the system, designed to solve the direct problem for patterns (i_F, i_{pk}), as well as index $A_F = \sum F_j$, consisting of columns $(i_{dk} \mid p_{vk})$ for different functions $f \in \mathcal{F}$.

In the initial state $A = \varnothing$, $m(i) = \varnothing$, $A_d = \varnothing$ and $A_F = \varnothing$.

Let us assume that $i_F \in U_F$ is a name by which a new function $f \in \mathcal{F}$ must be memorized.

First, direct problem is solved for all patterns $p_k = (i_{k1}, ..., i_{km}) \in X_f$, $k = 1, ..., l$, $l = |X_f|$. Intersection in the index A is calculated for each of them [3–5]:

$$\eta(A,\, p_k) = \bigcap_{j=1}^{m} a_{kj}\,,$$

where a_{kj} is the pattern of the column $(i_{kj} \mid a_{kj}) \in A_j$, i_{kj} is the name that is *j*-th coordinate of the pattern $p_k = (i_{k1}, \ldots, i_{km})$.

If $\eta(A,\, p_k) \neq \varnothing$ and exists at least one name $i_p \in \eta(A,\, p_k)$ which is such, that $m(i_p) = m$, then this name is unique and is the name under which pattern p_k is known.

In all other cases pattern p_k is an unknown new pattern, which must be memorized. In order to do this column engine chooses any pure name $i_{pk} \in U_p$, where U_p is the name domain for patterns p_k, and carries out additions:

$$A + (p_k \mid \{i_{pk}\}) = \{A_1 + (i_{k1} \mid \{i_{pk}\}), \ldots, A_m + (i_{km} \mid \{i_{pk}\})\}$$

,

where i_{kj} – name which is the *j*-th coordinate of the pattern p_k, $j = 1, \ldots, m$. Besides that, in the definition of the function $m(i_p)$ a pair (i_{pk}, m) is added. Name i_{pk} is a solution to a direct problem and is the name by which the pattern p_k will now be known [3–5].

After all the names i_{pk} are found, a direct problem is solved for l patterns (i_F, i_{pk}). Since f is a new function, all these patterns are new and they must be memorized. For each of them the column engine chooses pure name $i_{dk} \in U_d$, where U_d – name domain for patterns (i_F, i_{pk}), and carries out additions:

$$A_d + ((i_F, i_{pk}) \mid \{i_{dk}\}) = \{A_{d1} + (i_F \mid \{i_{dk}\}),\, A_{d2} + (i_{pk} \mid \{i_{dk}\})\}$$

, $k = 1, \ldots, l$.

At the same time, index A_F is added l columns $(i_{dk} \mid p_{vk})$, where p_{vk} is a value of the function f for pair (i_F, i_{pk}), known by the name i_{dk}:

$$A_F + \sum_{k=1}^{l} (i_{dk} \mid p_{vk})\,.$$

Similarly, all other new functions $f \in \mathcal{F}$ are memorized.

Assume now it's necessary to find the value of the function by the name $i_F \in U'_F$ for any pattern $p = (i_1, \ldots, i_m) \in P$, where $U'_F \subset U_F$ is a set of names of all the functions known to the system.

First a direct problem is solved for pattern p. For this an intersection $\eta(A,\, p)$ is calculated.

If $\eta(A,\, p) = \varnothing$ or $m(i) \neq m$ for $\forall i \in \eta(A,\, p)$, then pattern p is a new one and can't be included into the domain of any known functions.

If $\eta(A,\, p) \neq \varnothing$ and at least one name $i_p \in \eta(A,\, p)$ exists for which $m = m(i_p)$, then such name is a unique one and is a name, by which pattern p is known.

After that a direct problem is solved for pattern (i_F, i_p). If in the index A_d intersection $\eta(A_d, (i_F, i_p)) = \varnothing$, then pattern p doesn't belong to the domain of the function by the name i_F. If this intersection $\eta(A_d, (i_F, i_p)) \neq \varnothing$, it contains unique name i_d, which is a name of an pattern (i_F, i_p) [3–5]. Pattern a_{vd} of a column $(i_d \mid a_{vd}) \in A_F$ is a value of the function by the name i_F for pattern p.

Example. Assume there are two binary functions $f_1 \in \mathcal{F}(2, 3)$ and $f_2 \in \mathcal{F}(2, 1)$:

$$f_1 = \{((1,1), (1,2,3)), ((2,2), (2,3,1)), ((3,3), (3,1,2))\},$$

$$f_2 = \{((1,1), 3), ((2,1), 2), ((3,3), 3), ((3,1), 2)\}$$

After their memorizing under the names 1 and 2 index $A = \{A_1, A_2\}$, function $m(i)$, indices $A_d = \{A_{d1}, A_{d2}\}$ и A_F will look like this:

A_1

	4	5
1	2	3
1	2	3

A_2

5		
4		
1	2	3
1	2	3

$m(i)$

i	1	2	3	4	5
m	2	2	2	2	2

A_{d1}

	7
3	6
2	5
1	4
1	2

A_{d2}

4		6		
1	2	3	5	7
1	2	3	4	5

A_F

3	1	2				
2	3	1				
1	2	3	3	2	3	2
1	2	3	4	5	6	7

(Column patterns of the index A_F are vectors whose elements are written from bottom to top. In all other indexes, the column patterns are unordered sets of names.)

Assume it is necessary to find the value of the function by name $i_F = 1$ for pattern $p = (3, 2)$. Thus, for pattern p intersection $\eta(A,\, p) = \varnothing$, then pattern p is new and can't be included into the domains of the functions known to the system.

If we take pattern $p = (3, 1)$, then $\eta(A,\, p) = \{5\}$ and $m(5) = 2$, that is, pattern p is known by the name $i_p = 5$.

Consequently, pattern $(i_F, i_p) = (1, 5)$. For it intersection in the index A_d equals $\eta(A_d, (1, 5)) = \varnothing$, that is, pattern $(1, 5)$ is an unknown pattern. Consequently, pattern $p = (3, 1)$ is not included into the domain of the function by the name 1.

Finally, if we take pattern $p = (3, 3)$, then for it intersection $\eta(A, p) = \{3\}$ and $m(3) = 2$, that is, pattern p is known by the name $i_p = 3$. Pattern $(i_F, i_p) = (1, 3)$. For it intersection $\eta(A_d, (1, 3)) = \{3\}$, that is, pattern $(i_F, i_p) = (1, 3)$ is known by the name $i_d = 3$. A column with such name in index A_F is column $(3 \mid (3, 1, 2))$, that is, function by the name $i_F = 1$ for pattern $p = (3, 3)$ has a value $(3, 1, 2)$.

Similarly for the function by the name $i_F = 2$ and pattern $p = (2, 1)$ we obtain: pattern $p = (2, 1)$ is an pattern by the name 4, pattern $(i_F, i_p) = (2, 4)$ is known by the name 5, and pattern of the column by the name 5 in the index A_F equals 2. Consequently, the value of the function by the name $i_F = 2$ for pattern $p = (2, 1)$ equals 2.

VI. CHAIN OF FOLLOWING THE LINKS FOR SOLVING THE PROBLEM OF IMPLEMENTING FUNCTIONS

As already mentioned, the solution of basic problems can be considered as the implementation of following the link $p_i \to i$ in the direct problem and $i \to p_i$ in the inverse problem. This provides the basis for solving all other problems. The solution to any such problems is a chain of following the links until the result is received. It is easy to see that when solving the problem of implementing functions for $\forall i_F \in U'_F$ and $\forall p \in P$ the following the links chain consists of only three links – two direct problem links and one inverse problem link:

$$p \xrightarrow{\text{dir}} \begin{pmatrix} i_F \\ i_p \end{pmatrix} \xrightarrow{\text{dir}} i_d \xrightarrow{\text{inv}} p_v$$

Until now, it has been implicitly assumed that the memorizing of functions in column-based intelligent systems is performed before the system starts from pre-prepared data. However, the simplicity of the memorizing scheme allows you to form functions dynamically in the process of the system operation. As a result, as knowledge accumulates, such systems can simplify the scheme for solving existing problems, replacing long chains of links with chains of dynamically formed functions consisting of only three links.

VII. RESULTS

The sequence of equivalent function definitions proposed in this article, which results in representing the function as an index

$F = \{(i_{dk} \mid p_{vk}) \mid k = 1, \ldots, \mid X_f \mid\}$, provides a solution to the problem of implementing arbitrary

$f \in \mathcal{F}(n, m)$ functions in column-based intelligent systems. This sequence defines both the scheme for storing a new function and the scheme for obtaining function values. In the first two steps, these schemes do not differ. For function

$f : p \to p_v$ called i_F two direct tasks for the image are successively solved

p and image (i_F, i_p), where i_p – name of image p. Further, in the case of memorizing a function, to the index $A_F = \sum F_j$ column is added $(i_d \mid p_v)$, where i_d – image name (i_F, i_p), p_v – function value by name

i_F for image p. If the function value is received, the column is taken $(i_d \mid a_v) \in A_F$. Image of this column $a_v = p_v$ represents the value of a function by name i_F for image p.

Based on this, the ability of the systems under consideration to realize arbitrary functions was proved.

$f \in \mathcal{F}(n, m)$ using the elementwise comparison method and the intersection method. For each of these methods, the system used three indexes, one of which is the index

A_F.

In intelligent systems based on columns, the solution of basic problems serves as the basis for solving all other problems. The solution to any such problem is a chain of click-throughs until the result is received . When solving the problem of implementing the function, such a chain consists of only three links - two links of the direct problem

$p \to i_p$, $(i_F, i_p) \to i_d$ and one link inverse task

$i_d \to p_v$.

The simplicity of the memorizing scheme allows you to form functions dynamically in the process of the system operation. As a result, as knowledge accumulates, such systems can simplify the scheme for solving existing problems, replacing long chains of links with chains of dynamically formed functions consisting of only three links.

VIII. CONCLUSION

Column-based intelligent systems are an example of systems whose primary function is pattern memorizing. Such systems are not widespread since at first glance it seems that such systems, in addition to memorizing patterns, cannot solve any other problems. Column-based intelligent systems prove that this is not the case. Despite its extreme simplicity, column-based intelligent systems demonstrate a wide range of solutions to various problems. As was shown in previous works, such systems can work under incomplete information and have the ability to make predictions. It was proved that column-based intelligent systems can solve classification problems, which make up a significant part of the problems of artificial intelligence. At the same time, the chain of following the links for solving the classification problem consists of only two links - one direct problem link and one inverse problem link.

In addition, it was proved that there is the possibility of implementing Boolean functions $f : B^n \to B$, where $B = \{0, 1\}$. This result demonstrated the possibility of solving with the help of the systems under consideration those problems for which a logical approach can be applied.

In this paper, for column-based intelligent systems, the more complex problem of realizing arbitrary functions $f : U^n \to U^m$ was considered. It was shown that when solving the implementation problem, i.e. finding the value of the function, the chain of following the links consists of only three links - two direct problem links and one inverse problem link. Moreover, the memorization of new functions can be performed not only before the start of the system from pre-prepared data. The simplicity of the memorization circuit allows the formation of functions dynamically during the operation of the system. As a result, with the accumulation of knowledge, the considered systems can simplify the scheme for solving existing problems by replacing long chains of following the links, for example, logical inference chains, with chains of dynamically formed functions consisting of only three links. The combination in one system of the ability to predict and work under incomplete information, the ability to solve classification problems, realize complex functions and learning as knowledge accumulates - all this demonstrates both the versatility of column-based intelligent systems, and the possibility of their use for solutions to various problems.

REFERENCES

1. Mikhailov A.M. Pattern recognition by indexing // Automation and Remote Control, 2012, Vol. 73, No. 4, pp. 717–724.
2. Mikhailov A.M. An indexing-based approach to pattern and video clip recognition // Automation and Remote Control, 2014, Vol. 75, No. 12, pp. 2201–2211.
3. Chesnokov A.M. Column-Based Intelligent Systems (Intellektual'nye sistemy na osnove kolonok) // Upravlenie bol'shimi sistemami (Large-Scale Systems Control), 2013, No. 46, pp. 118–146.
4. Chesnokov A.M. Column-Based Intelligent Systems under Incomplete Information (Intellektual'nye sistemy na osnove kolonok pri nepolnoy informatsii) // Upravlenie bol'shimi sistemami (Large-Scale Systems Control), 2014,No. 50, pp. 84–98.
5. Chesnokov A.M. Vvedenie v obshchuyu teoriyu kolonok (Introduction to General Columns Theory). – M.: IPU RAN publ., 2012.
6. Chesnokov A.M. Finite Multisets as Patterns in Column-Based Intelligent Systems // Automation and Remote Control, 2015, Vol. 76, No. 9, pp. 1681–1688.
7. Mikhailov A., Pok Y.M. Artificial Neural Cortex // Proceedings of Artificial Neural Networks in Engineering Conference (ANNIE 2001), Nov. 4−7, 2001, St. Louis, Missouri, U.S.A.
8. Mikhailov A. Biologically Inspired Artificial Neural Cortex and its Formalism // World Academy of Science, Engineering and Technology, August 2009, Vol. 56, p. 121.
9. Mikhailov A. Indexing-based Pattern Recognition // Advanced Materials Research. – 2012, Vols. 403–408, pp. 5254–5259.