system [13]. In the process of *versioning of the knowledge base fragment*, a tree-like linear structure of states of this knowledge base fragment (**tree of states of knowledge base fragments**) and a tree-like linear structure of state identifiers of this knowledge base fragment (**tree of state identifiers of knowledge base fragments**) are built. Both structures are represented in the SC-code.

The Subsystem for versioning of knowledge base fragments consists of the following components:

- *specification for the versioning model of knowledge base fragments* (i.e., documentation (knowledge bases) describing methods, tools, and algorithms of *versioning of knowledge base fragment*);
- *problem solver* [14], including:
  - an *Abstract sc-agent for generating the initial state of a knowledge base fragment in its state stack*;
  - an *Abstract sc-agent for integrating changes with the current state of a knowledge base fragment and adding the resulting state of the knowledge base fragment to its state stack*;
  - an *Abstract sc-agent for identifying the state of the knowledge base fragment in the stack of state identifiers of a given knowledge base fragment*;
  - an *Abstract sc-agent for getting the state of a knowledge base fragment by its identifier*;
  - an *Abstract sc-agent for getting the version of a knowledge base fragment by its identifier*.
- *program interface* and *user interface* to use the *versioning of knowledge base fragments*.

The *state of knowledge base fragments* is pairs of two sets. The elements of the first set are the membership arcs of the certain substructures for decomposing the more general structure, which in the version of this *state of the knowledge base fragment* have not been changed. The elements of the second set are the membership arcs of the certain substructures for decomposing the more general structure, which have been changed in this version. Both sets can be represented as a role relation (attribute set), denoting which and in what way the certain structures of the general structure of states are represented. These two sets play an important role in the operational semantics used when implementing agents to work with the history of structure changes.

Reconstructing a version by the *state of the knowledge base fragment* takes only one traversal of the subtree, i.e., a traversal of this state. Structures located in the hierarchy lower than the structure, which includes a membership arc of the composition of a more general structure, which in turn is an element in the attribute set of membership arcs of changed (unchanged) structures, are considered to be changed (unchanged) until an arc from the second attribute set is encountered, and those above the arc are considered to be unchanged (changed) until an arc from the first attribute set is encountered. This approach is easy to implement and does not require a large number of copies for bindings of decomposition and membership arcs for changed structures, compared to the previous cases.

Let us consider an example of versioning a particular neural network method represented in *Neuro-SCP* and described in [10]. This method solves the classical "EXCLUSIVE OR" problem [15].

In *Neuro-SCP*, the implementation of this ANN is reduced to a single layer interpretation operator, for which the *matrix of synapse weights*, *threshold*, and *activation function* are defined. In Figure 1, the *neuro-SCP* interpretation operator of the *layer of a single-layer perseptron*, solving the "EXCLUSIVE OR" problem, is shown.

In Figure 2, the *single-layer perseptron scheme*, which solves this problem and from which the *Neuro-SCP* operator has been described, is demonstrated.

Any of these elements can be changed in the process of training and reconfiguration of the *ANN*. Within the proposed approach, such *ANN* will correspond to its own *tree of states*. Let us suppose that a given *ANN* has been trained and its *synaptic weight matrix* of a single *layer* has changed.

In Figure 3, a *tree of states* that stores two versions of the same *ANN* — before training and after training — is shown. The root of this tree is an instance of the history class. There are two states associated with the root, each of which is an oriented pair of sets.

For each state, the time of the beginning of the state existence is set. All elements describing the states are *temporal*, since they exist temporarily and each state can be replaced by another.

For the first state, the first set was empty, since it is the first state in the tree and it cannot refer to previous states. The second set considered above describes all elements that were added in this version. Since it is the first state, this set describes all membership arcs involved in the description of the operator.

The first set of the second state — the set of elements that have not changed from the previous state — is described using the constructions involved in describing the second set of the first state. The second set of the second state, on the other hand, describes only the changes or, to be more precise, only the membership arcs that have been added in the new state. Thus, it is possible to understand that the weight matrix has changed in the new state. It can be assumed that the *ANN* has been retrained.

The use of ANNs as a problem-solving method implies the use of an already designed and trained ANNs. However, the presence of a neural network method description language in ostis-system memory opens the way for automation of the processes of designing and training ANNs. Such automation is represented by separate classes of problems and the corresponding skills for their solution.

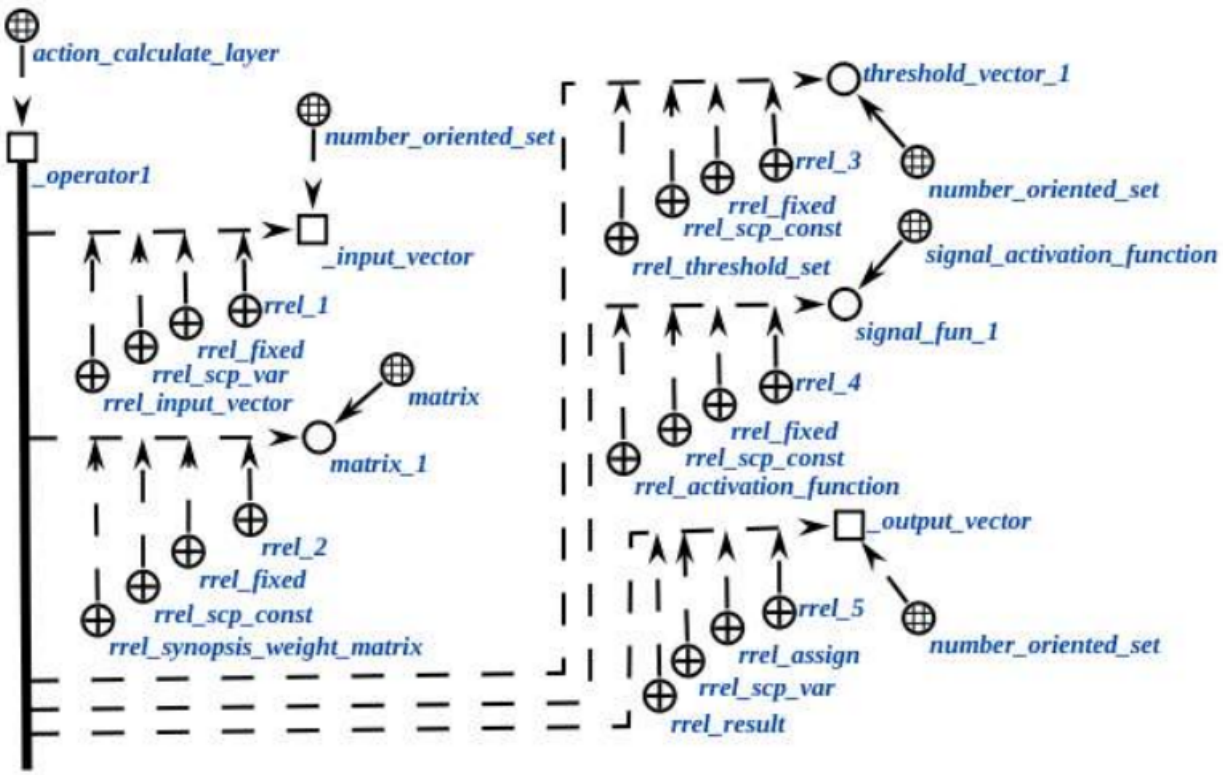

Figure 1: Operator for interpreting the ANN layer on Neuro-SCP

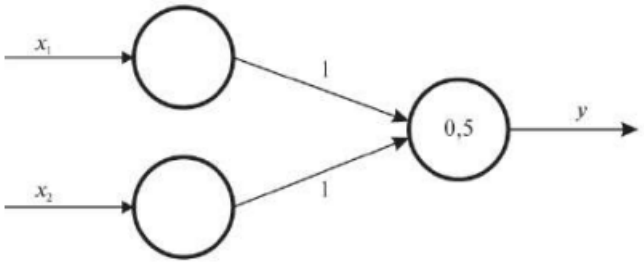

Figure 2. Scheme of a single-layer perseptron solving the "EXCLUSIVE OR" problem [15]

III. CONCLUSION

The described versioning model allows storing and restoring any state not only of the *neural network method* described with *Neuro-SCP* in the *knowledge base* but also any *knowledge base fragment* in principle with minimal effort.

The model can be used to describe not only what has been changed but also at what time and under what influences. If necessary, each state can be described from various angles — efficiency, validity, operational history, etc.

In the case of ***convergence and integration of ANNs with knowledge bases of intelligent systems***, such a model will allow considering ANNs not only as a problem-solving method but also as a design object. The automation of such design will become a task that can be posed by the same intelligent system, in which the neural network is described and interpreted.

The availability of unification of the knowledge representation describing the problem and the methods of solving these problems allows the whole arsenal of *problem-solving methods of the intelligent system* to be used for design. Such methods can achieve good results in design automation by accessing the knowledge describing the problem to be solved by the designed *ANNs*, the context of the problem to be solved, the history of the intelligent system solving similar problems, etc.

A further development of this work is the design and implementation of *intelligent design framework of ANNs*, which will automate various activities for ANNs designers.

ACKNOWLEDGMENT

The author would like to thank the research groups of the Departments of Intelligent Information Technologies of the Belarusian State University of Informatics and Radioelectronics and the Brest State Technical University for their help in the work and valuable comments, in particular, Vladimir Golenkov, Vladimir Golovko, Aliaksandr Kroshchanka, and Nikita Zotov.

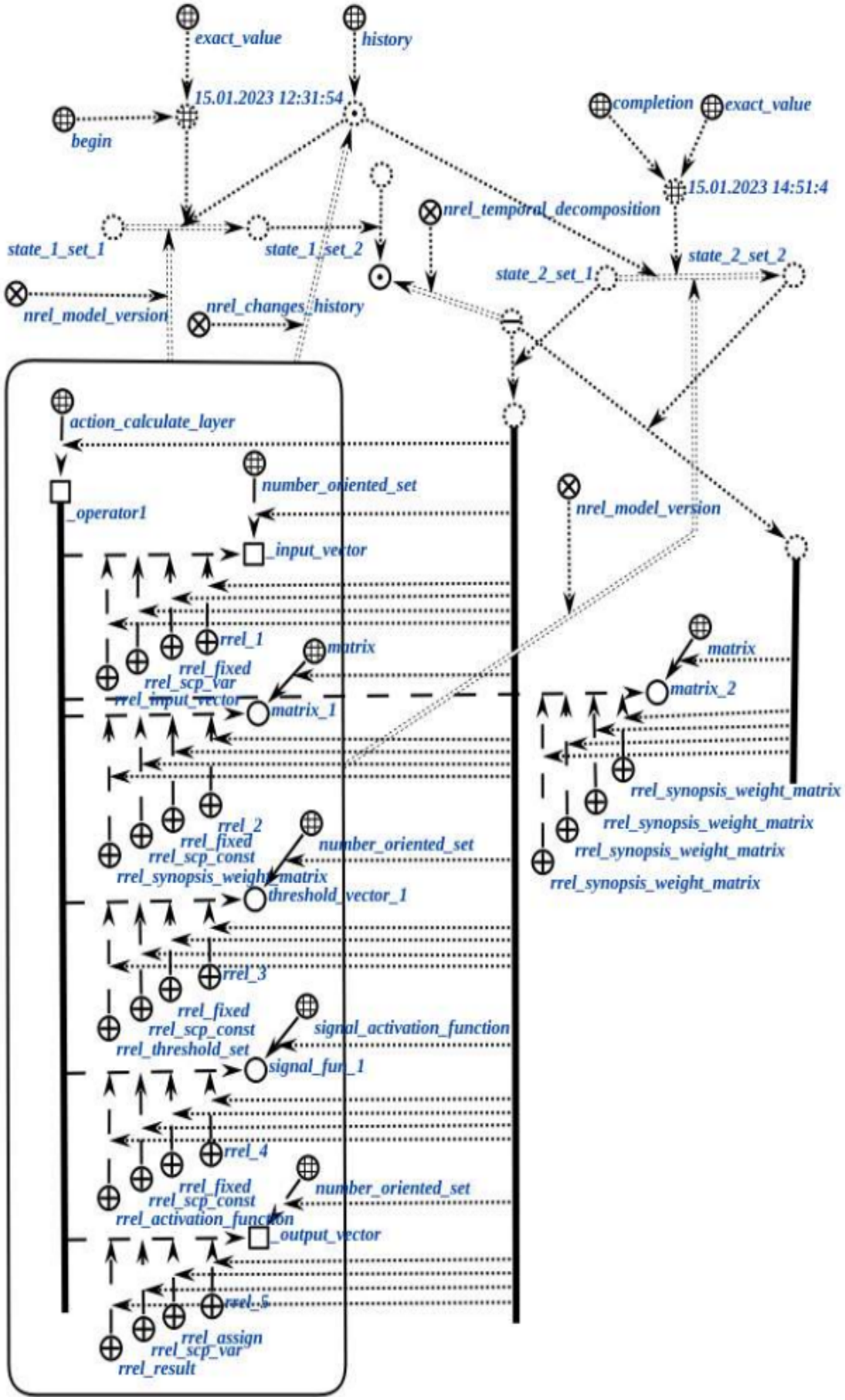

Figure 3. Example of an ANN tree of states