of three-element constructions of connectivity components in the specified graph template

- *set of indices of the highest-priority three-element constructions for the search for program objects of connectivity components in the specified graph template*
- *tuple of sets of sc-addresses of sc-memory elements corresponding to sc-connectors, such three-element constructions that are not isomorphic to the corresponding three-element constructions of the specified graph template, whose indices are equal to the position indices of sets in this oriented set*
- *tuple of sets of sc-addresses of sc-memory elements corresponding to sc-connectors of such three-element constructions that are isomorphic to the corresponding three-element constructions of the specified graph template, whose indices are equal to the position indices of the sets in this oriented set*
- *tuple of sets of sc-addresses of sc-memory elements corresponding to sc-connectors that are the second elements of the corresponding three-element constructions of found sc-memory constructions isomorphic to the specified graph template, whose indices are equal to the position indices of sets in this oriented set*
- *tuple of sets of indices of program objects of three-element constructions of the specified graph template for constructions isomorphic to it found from in sc-memory, whose indices are equal to the position indices of sets in this oriented set*
- *index of the last found sc-memory construction according to the specified graph template*
- *set of indices of all found and isomorphic constructions in sc-memory according to the specified graph template*

}
}
}

The graph template preprocessing step consists of the following intermediate processing steps:

- Addition to the relation *local identifier of some element in some program object of a three-element construction of the specified graph template and the set of all indices of program objects of three- element constructions in the specified graph template with this element** of all pairs with local element identifiers in some program object of a three-element construction of the specified graph template and sets of indices of corresponding program objects of three-element constructions in the specified graph template. The sets of such program objects do not include those *program objects of three-element constructions* in the specified graph template, whose indices are included in the local identifier of the element itself. This element's local identifier is not the element's local identifier within the entire graph template. Such a local identifier is formed by the system itself, and not by the user of the *Method of adding a three-element construction for the specified graph template (Method of adding a five-element construction for the specified reference graph)* and consists of a local element identifier within the entire graph template and the number of the corresponding program object of the three-element structure in the specified graph template. Knowing such a local identifier of an element of *some program object of a three-element construction* of the graph template,one can quickly access other *program objects of three-element constructions* that contain this element.
- Removal from the sets of indices of *program objects of three-element constructions*, which are the second components of pairs of the relation *local identifier of some element in some program object of a three-element construction of the specified graph template and the set of all indices of program objects of three-element constructions in the specified graph template with this element** and in which there are elements that are the first components of these pairs, all such indices of *program objects of three-element constructions*, passing through which in the process of searching for constructions isomorphic according to the specified graph template can lead to a looping of the search algorithm. This pre-processing stage of the formed *graph template program object* makes it possible to eliminate transitions along such *program objects of three-element constructions* of the specified graph template as much as possible, which significantly complicating the process of isomorphic search for constructions according to the specified graph template. Since SC-code itself allows one to represent constructions of any possible configuration, it is impossible to say exactly which configurations of constructions can lead to cyclic situations when the isomorphic search algorithm for these structures is executed according to the specified graph template. A more universal algorithm for eliminating loops in the graph template can lead to significant additional time costs, since it may require a deeper syntactic analysis in the original graph template, so it is recommended to implement the conditions by which you can determine the program objects of three-element constructions, the transition to which can lead to cyclic situations in the processing of the graph template. The elimination of loops in the

graph template allows the algorithm of isomorphic search for structures on the specified graph template to more efficiently perform all the required operations on graphs, therefore it is a key step in preprocessing the original graph template program object. Also, this step cannot be performed together with the previous step, since in order to eliminate all loops in the graph template, it is necessary to know completely all possible transitions along this graph template.

- Search for all connectivity components in the specified graph template, that is, unrelated subgraphs in this graph, and add all program objects of three-element constructions corresponding to these connected components to the tuples of sets of numbers of program objects of three-element constructions connectivity components in the specified graph template. Thus, this makes it possible to find even such connected components that could be obtained after performing the second step of the algorithm for preprocessing the specified graph template, that is, eliminating cycles in the specified graph template. Dividing a graph template into connectivity components could be one of the solutions to the problem of eliminating a cycle in the specified graph template, however, the algorithm for isomorphic search for structures on the specified graph template is more advanced and allows one to find all three-element constructions for topics of three-element graph template constructions that have the same first or third element, so splitting the graph template into connected components is not used in the previous step
- The last step of the *graph template preprocessing stage* is to select the connectivity components found at the previous *Pre-processing stage of graph template of the highest-priority program objects of three-element constructions* in the specified graph template. The highest priority program object of the three-element design is the object with a priority number equal to zero, the most non-priority object is the object with a priority number equal to six. intelligence, since the maximum level of cybernetic system processor quality in terms of the variety of problem-solving models interpreted by the processor of a cybernetic system is its universality, that is, its models and various knowledge types is carried out. Due to the fact three-element In this case, if there are several *program objects of three-element constructions* that have the same priority number, then the object with the first (third) element, which is the sc-address of the sc-memory element, is considered to have the highest priority. has the least number of elements corresponding to outgoing (incoming) sc-connectors. As a result, *a set of numbers of the highest-priority three-element constructions for the search for program objects of the connectivity components in the specified graph template* is formed.

Thus, graph template preprocessing stage makes it possible to significantly simplify the processing of a graph template at the stage of searching for constructions isomorphic to it. The next **stage of searching for scmemory constructions isomorphic to the specified graph template** includes the following steps:

- If the specified tuple of *program objects of sc-memory constructions isomorphic to the specified graph template* is not empty, then delete all program objects of constructions in sc-memory from it.
- If *the set of numbers of the highest-priority three-element constructions for searching program objects of connectivity components in the specified graph template* is empty, then this means that the specified *graph template* is empty. In this case, the result of the search is an empty *tuple of program objects of sc-memory constructions isomorphic to the specified graph template and The stage of searching for scmemory constructions isomorphic to the specified graph template* ends with a successful result
- Initialize *number of the last found sc-memory construction according to the specified graph template* with a value equal to zero. Set *Number of the current found sc-memory construction according to the specified graph template equal to number of the last found sc-memory construction according to the specified graph template.* Set *set of numbers "equivalent" program objects of constructions to* be equal to set of numbers of the highest priority for searching program objects of three-element constructions of connectivity components in the specified graph template. Set *set of numbers of current program objects of constructions equal to set of numbers of "equivalent" program objects of constructions.*
- Select the next number from *set of numbers "equivalent" program objects of constructions.* According to the *received number from tuple of program objects of three-element constructions in the specified graph template take the corresponding program object of three-element construction of the specified graph template* in this graph template.
- For the *selected program object of a three-element construction* in the specified graph template, find all such *program objects of three-element constructions*, (1) whose elements have the specified classes and sc-addresses the same as the classes and sc-addresses of the elements of the selected program object of the three-element construction, respectively, while either the first or third of their elements have the same local identifiers in the specified graph template or do not have them at all, (2) for which the corresponding replacements were not found, that is, the set located in the *tuple of sets of numbers of program objects of three-element constructions of the specified graph template for isomorphic constructions found using it in sc-memory, whose numbers are equal to the position numbers of sets in this oriented set* by *the number of the current found sc-memory construc-*

tion according to the specified graph template, the numbers of the found program objects of three-element constructions do not belong, and also the numbers of which do not belong to the set of numbers of the current program objects of the constructions.

- If the received *set of numbers of "equivalent" program objects of constructions is empty*, then *terminate this iteration of the algorithm.*
- If the received *set of numbers "equivalent" program objects of constructions* is not empty, then choose a random number from the *set of numbers "equivalent" program objects of constructions.* According to the received number from *tuple of program objects of three-element constructions in the specified graph template* take the corresponding *program object of three-element construction of the specified graph template* in this graph template.
- Based on the obtained *program object of a three-element construction, create a an iterator for searching for three-element sc-memory construction. Creation of iterator for searching three-element sc-memory construction is done using the Method of creating iterator for searching three-element sc-memory construction.* The parameters of the f three-element constructions f three-element constructions f three-element constructions f three-element constructions f three-element constructions f three-element constructions f three-element constructions f three-element constructions f three-element constructions f three-element constructions f three-element constructions f three-element constructions f three-asaaasdasdsdaa asdasdsad asdasds method are assigned to the elements of the specified *program object of the three-element construction*, while instead of *classes of sc-memory elements corresponding to sc-variables (classes of sc-memory elements corresponding to sc-metavariables)*, the corresponding them *classes of sc-memory elements corresponding to sc-constants (classes of sc-memory elements corresponding to sc-variables).* That is, for example, if some *program object of a three-element construction element* has a *class of sc-memory elements corresponding to variable sc-nodes*, then instead of it for the *Method of creating a three-element sc-memory construction search iterator*, the corresponding *class of sc-memory elements corresponding to constant sc-nodes* is used. Thus, when passing from *program object of a three-element construction to program three-element sc-memory construction search iterator*, the degree of variability of the elements of program *object of a three-element construction decreases: classes of sc-memory elements corresponding to sc-metavariables* are converted *to classes of sc-memory elements corresponding to sc-variables, and classes of sc-memory elements corresponding to sc-variables — to classes of sc-memory elements corresponding to sc-constants. If class of element of program object of three-element construction is not a non-strict subset of class of sc-memory elements corresponding to sc-metavariables or class of sc-memory elements corresponding to sc-variables*, then the degree of variability is not reduced .
- Using the *Method of moving to the next sc-memory construction "suitable" for the specified iterator go to the sc-memory construction* isomorphic to the specified construction in the graph template.

Despite the wide range of tasks that can be performed using the current implementation of isomorphic search, there are a number of reasons why this and other implementations of isomorphic search should not be used.

- *isomorphic search allows* you to solve a wide range of problems if all knowledge isomorphic to the specified *graph template is in the knowledge base or is missing.* That is, the quality level of *isomorphic search* directly depends on the state of knowledge base. The more diverse *knowledge base* fragments are, the worse the performance of isomorphic search is;
- The cost of *searching for large graph templates* may be at odds with the desires of the developer or user. The larger the *graph template*, the more situations in which the *isomorphic search* algorithm can behave abnormally
- Most of the problems solved with *isomorphic search* can and should be solved with *three- and five-element search iterators.* The simpler the method of solving problems, the fewer errors and emergency situations you can get.

VI. ADVANTAGES AND DISADVANTAGES OF THE IMPLEMENTATION OF THE INFORMATION RETRIEVAL SUBSYSTEM IN THE OSTIS-PLATFORM

The current *Software interface of Implementation of sc-memory in the ostis-platform allows*:

- Implementing platform-specific subsystems of the current *software implementation of the ostis-platform* to the extent necessary and sufficient, practically independently of the *Implementation of sc-memory in the ostis-platform.* That is, the current *Software interface of Implementation of sc-memory in the ostis-platform* is a way to unify access to the software *Implementation of sc-memory in the ostis-platform* and allows easily to replace various implementations of sc-memory with *method representation language C++, while the Software interface of Implementation of sc-memory in the ostis-platform itself practically*
  does not change or does not change at all.
- Implementing basic tools for designing platform-independent ostis systems, e.g. *Implementation of scp-interpreter.*
- Generating and expanding the *Library of reusable components of Software implementation of the ostis- platform* with components that use the methods of *Implementation of the sc-memory in the*

ostis-platform and are part of various plug-ins of the current *Software interface of Implementation of scmemory in the ostis-platform*.
- Providing different levels of access to *Implementation of sc-memory in the ostis-platform*, including the levels of access for different users of the *Software implementation of the ostis-platform*.

It is worth noting that *Software interface of Implementation of sc-memory in ostis-platform* cannot exist separately from the current *Implementation of scmemory in the ostis-platform*. In addition, it is part of the Implementation of sc-memory in the ostis-platform, that is, it is designed Method of obtaining the sc-address of an element of a program object of an sc-memory construn and developed in accordance with the implementation of the sc-memory itself. However, if necessary, it can be used for various modifications or versions of the current *Implementation of sc-memory in the ostis-platform*.

Despite the wide range of functionality of the current Software interface of *Implementation of sc-memory in the ostis-platform*, its disadvantages include the following:

- At the level of the *Software interface of Implementation of sc-memory in the ostis-platform*, there is no limit to the range of classes of sc-elements in scmemory that can be set as arguments, for example, to the *Method of creating an sc-memory element of a given class, corresponding to an sc-node* and *Method of creating an sc-memory element of a given class, corresponding to some sc-connector*.
- Due to shortcomings in the current implementation of the agent architecture in the *software implementation of the ostis-platform* it is impossible for the *Software interface of Implementation of sc-memory in the ostis-platform* to use *Implementation of scmemory in ostis-platform* stored as a compiled file. First of all, this is due to the fact that platform-specific agents are implemented by means that utilize creation of source files when building the entire platform. Thus compiled files remain dependent on the device where they were built.

In general, isomorphic search can be a useful tool in theoretical studies and some specialized applications, but in most cases there are better ways to work with graphs.

VII. CONCLUSION

Let us list the main ideas of this work:

- to solve information retrieval tasks in ostis-systems, the *Implementation of the information retrieval subsystem of the current Software implementation of the ostis-platform* is used;
- *Implementation of the information retrieval subsystem in the Software implementation of the ostis-platform* has a software interface that can be used in any platform-dependent component (subsystem);
- *Implementation of the information retrieval subsystem in the Software implementation of the ostis-platform* includes iterative methods for searching for sc-memory constructions and methods for searching for sc-memory constructions according to the specified graph template;
- to solve most information retrieval problems, it is sufficient to use iterative methods for searching for sc-memory constructions;
- the current implementation of isomorphic search is not universal and is limited to a certain set of graph templates, and also strongly depends on the state of the knowledge base.

When designing graph templates in one of the languages of the external representation of SC-code [20], one should:

- Minimize the number of cycles by splitting, for example, key constant sets into subsets that are not interconnected in this graph template. If the cycle in the graph cannot be eliminated, then leave it as it is, or reconsider the original problem for the possibility of simplifying its solution.
- Select among all those sc-constructions that can be selected by the search procedure as the first sc-construction, only the one that simplifies the work of the search procedure as much as possible for the specified one in the subject domain.
- Minimize the number of sc-constructions, the removal of which does not change the meaning of the found constructions and/or can be specified/checked later (for example, when the entity is already found and the class membership can be checked later) and/or the removal of which simplifies the choice of path search in a graph isomorphic to the specified graph template.

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.

References

[1] D. Shunkevich, D. Koronchik, "Ontological approach to the development of a software model of a semantic computer based on the traditional computer architecture," Otkrytye semanticheskie tekhnologii proektirovaniya intellektual'nykh system [Open semantic technologies for intelligent systems], pp. 75–92, 2021.

[2] N. Zotov, "Software platform for next-generation intelligent computer systems," Otkrytye semanticheskie tekhnologii proektirovaniya intellektual'nykh system [Open semantic technologies for intelligent systems], pp. 297-326, 2022.

[3] A. Zagorskiy, "Factors that determine the level of intelligence of cybernetic systems," Otkrytye semanticheskie tekhnologii proektirovaniya intellektual'nykh system [Open semantic technologies for intelligent systems], p. 13-26, 2022.