retrieval methods of Implementation of sc-memory in the ostis-platform

three- and five-element sc-memory construction search iterator

⊂ *software object*
:= [ScIterator]
∈ *C++*

Software interface for three- and five-element sc-memory construction search iterator

⊃=
{
⇐ *software interface*:*
three- and five-element sc-memory construction search iterator

Method of moving to the next sc-memory construction "suitable" for the specified iterator

∈ *method*
⇒ *method header in method representation language*:*
[bool Next() const]
∈ *C++*
⇒ *method result class*:*
• *boolean*

Method of accessing the sc-address of an element of the specified sc-memory construction by the position index of this element in the specified construction

∈ method
⇒ *method header in method representation language*:*
[ScAddr Get(size_t idx) const]
∈ *C++*
⇒ *method input argument classes*:*
⟨• *32-bit integer*
⟩
⇒ *method result class*:*
• *sc-address of sc-memory element*
⇒ *class of exceptions*:*
• *invalid element position in the specified sc-memory construction*

}
}

For *Method for creating a three-element sc-memory construction search iterator*, as well as for *Method for creating a five-element sc-memory construction search iterator*, various combinations of parameters can be used, except for combinations where all parameters are classes of *sc-memory elements*. For simplicity and compactness of the terms used at the level of implementation of methods for creating iterators for searching for structures in sc-memory, additional notations are introduced: the symbol "f" (from the English word "fixed") denotes the fact that the parameter of a given method for creating an *sc-memory construction* search iterator is the *sc-address* of some *sc-memory element*, and the symbol "a" (from the English word "assign") denotes *sc-memory element class^*. For *Method for creating a three-element sc-memory construction search iterator*, the correct designation of the desired constructions will be a three character long combination of characters "f" and "a", and for *Method for creating a five-element sc-memory construction search iterator* — a five-character combination of "f" and "a". In the *SCP* Language, to indicate whether a variable has the specified value, the corresponding role relations are used: for variables of class "f" — *scp-operand with the specified value'*, for variables of class "a" — *scp-operand with free value'* [17].

Software interface for information retrieval methods in the Implementation of sc-memory in the ostis-platform

⊃=
{

Method for creating a three-element sc-memory construction search iterator

⊃ *Method for creating an fff-construction search iterator*
∈ *method*
⇒ *method input argument classes*:*
⟨• *sc-address of sc-memory element*
• *sc-address of sc-memory element*
• *sc-address of sc-memory element*
⟩
⊃ *Method for creating an faa-construction search iterator*
∈ *method*
⇒ *method input argument classes*:*
⟨• *sc-address of sc-memory element*
• *sc-memory element class^*
• *sc-memory element class^*
⟩
⊃ *Method for creating an aaf-construction search iterator*
∈ *method*
⇒ *method input argument classes*:*
⟨• *sc-memory element class^*
• *sc-memory element class^*
• *sc-address of sc-memory element*
⟩
⊃ *Method for creating an faf-construction search iterator*
∈ *method*
⇒ *method input argument classes*:*
⟨• *sc-address of sc-memory element*
• *sc-memory element class^*
• *sc-address of sc-memory element*
⟩
⊃ *Method for creating an afa-construction search iterator*
∈ *method*
⇒ *method input argument classes*:*

⟨• *sc-memory element class*ˆ
• *sc-address of sc-memory element*
• *sc-memory element class*ˆ
⟩

}

These variants of the implementation of the *Method for creating a three-element sc-memory construction search iterator* are sufficient for solving any search and navigation tasks. ***Method for creating an ffa-construction search iterator*** and ***Method for creating an aff-construction search iterator*** are possible, but in practice there is no need to look for a third sc-memory element by the known element corresponding to the sc-connector and the element corresponding to the sc-element from which this sc-connector exits or into which this sc-connector enters. Such a problem can be solved using ***Method for creating an afa-construction search iterator***. However, the implementation of *Method for creating a five-element sc-memory construction search iterator* is not at all necessary, since all tasks solved using this method can also be solved using *Method for creating a thre-eelement sc-memory construction search iterator*, however, the implementation of *Method for creating a five-element sc-memory construction search iterator* allows you to make the text of the method more compact compared to the method that would use *Method for creating a three-element sc-memory construction search iterator.*

The following can be specified as all three arguments for the *Method for creating a three-element sc-memory construction search iterator*:

- *sc-addresses of sc-memory elements* (for example, when solving the problem of checking the incidence of all three specified sc-memory elements),
- *sc-storage element address, class of sc-storage elements corresponding to sc-connectors*ˆthat come out of the *sc-storage element* passed as the first argument, and *sc-address of the sc-memory element* corresponding to some *sc-element* that contains the required *sc-connectors* (for example, when solving the problem of finding all *sc-memory elements corresponding to sc-connectors between sc-elements* for which the specified *sc-memory elements* correspond),
- *sc-storage element address, class of sc-storage elements corresponding to sc-connectors*ˆthat come out of the *sc-storage element* passed as the first argument, and *class of sc-memory elements corresponding to some sc-elements*; which include the required *sc-connectors* (for example, when solving the problem of finding all sc-memory elements that *correspond to sc-connectors* coming from the *sc-element* that matches the *sc-memory element* passed as the first argument),
- *class of sc-memory sc-elements; class of sc-memory elements corresponding to sc-connectors*ˆ that come out of the *sc-memory elements specified as the first argument, and sc-address of the sc-memory element corresponding to some sc-element*, which contains the required *sc-connectors* (for example, when solving the problem of finding all *sc-memory elements corresponding to sc-connectors*, contained in the *sc-element* that matches the *sc-memory element* specified as the third argument),
- *sc-memory element class; sc-address of the sc-memory element corresponding to the sc-connector* that comes out of the *sc-memory element* passed as the first argument, and *class of sc-memory elements corresponding to some sc-element*; which contains the required *sc-connector* (for example, the task of finding *sc-memory elements corresponding to sc-elements*, one of which is the *sc-element* from which the *sc-connector* emerges, for which the specified *sc-memory element* matches, and the second of which is the *sc-element* that includes this *sc-connector* for which the specified *sc-memory element* matches)
- and so on.

As all five arguments for the ***Method for creating a five-element sc-memory construction search iterator***, other combinations can be specified that are not specified in the presented classification. However, this is not necessary, since all tasks solved using such *iterators* can be solved by already existing *five-element sc-memory construction search iterators.*

V. Implementation of isomorphic search for sc-memory constructions of the ostis-platform according to the specified graph template

Method for creating a three-element sc-memory construction search iterator and *Method for creating a fiveelement sc-memory construction search iterator*, as well as *Software interface for three- and five-element sc-memory construction search iterator* are quite powerful tools for solving any *information retrieval problems* in applied *ostis-systems.* For example, in *inference* [18] problems, it is considered convenient to solve *problems* when *search for structures* of any necessary *configuration in sc-memory* reduces to ***isomorphic search*** of these constructions according to the specified ***graph template*** (Figure. 3). Such graph templates can be any *atomic logical formulas* included in any other *non-atomic formula* [18].

Isomorphic search is one way to solve the *problem* of finding a subgraph in a *graph* (see [21]). The problem consists of finding all occurrences of the specified *graph template* in the source graph. The *isomorphic search* process can be implemented using various algorithms. One of them is *Ullman's Algorithm* [22], which is based on using an adjacency matrix to determine the correspondence between graph vertices. Another algorithm is the *VF2 Algorithm*, which uses a comparison function to check if the corresponding nodes and edges in two graphs match (see [23]). In modern computer science, there

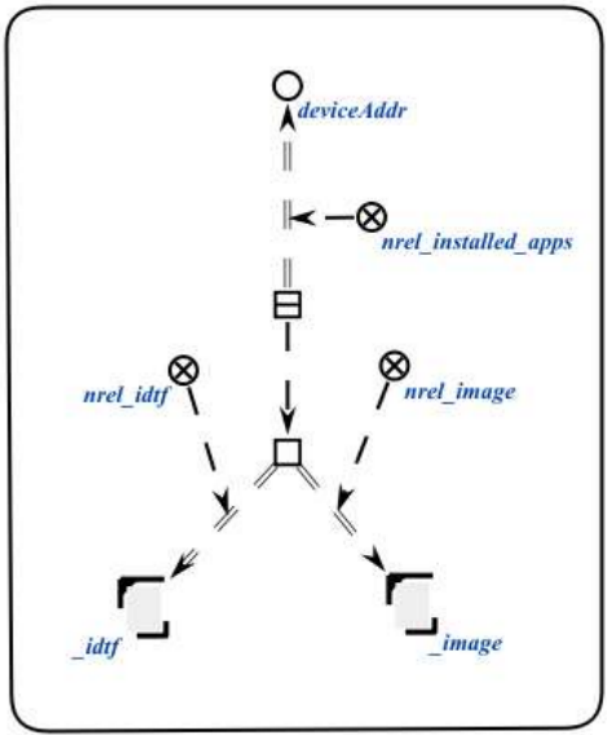

Figure 3. SC.g-text. Graph template example

are algorithms that allow *solving the problem of isomorphic search* in subexponential time (see [24]).

The fundamental principle of the *OSTIS Technology* that is currently under development is the principle of adopting the best existing technologies for the development of *ostis-systems* [25]. However, due to various circumstances, for example, connected with the specific features of the *Implementation of sc-memory in the ostis platform*, as well as the requirements imposed on sc-agents involved in logical inference, it is necessary to apply and test new solutions. Within the framework of the current *Implementation of sc-memory in the ostis-platform*, a concept of *isomorphic search* has been developed, which allows to find graphs isomorphic to fragments of a given *graph template in optimal time.*

In general there is no need to implement *isomorphic search* in a generic way. This is explained by the following:

- *isomorphic search* is an NP-complete problem, which means that the cost of solving it grows exponentially with the size of the input data, and there is no efficient algorithm for solving the *isomorphic search* problem yet;
- as a result of determining the isomorphism of two given graphs, several nodes can be found that correspond to each other, but are not actually isomorphic;
- due to the exponential growth in the number of possible isomorphism variants with increasing graph size, even small errors in the calculation of isomorphism can lead to severe distortion of the results;
- for large graphs, the time spent on enumeration of all possible isomorphisms can be very high. This can reduce search efficiency and limit the use of *isomorphic search* in real-world problems (see [26]);
- search complexity increases with the number of loops in the original *graph*, as it results in more iterations;
- existing algorithms are either slow or waste memory, resulting in *isomorphic search* being slow.

Some isomorphic search algorithms even have *O(n!)* complexity and cannot be used for large graphs (see [21]). Despite all the problems associated with *isomorphic search*, for the convenience of solving logical problems, the current *Software implementation of the ostis-platform* implements the "most appropriate" *isomorphic search* algorithm. The current version of *isomorphic search* is implemented in the *Method for finding sc-memory constructions isomorphic to the specified graph template.* This method allows you to find *sc-memory constructions* that are isomorphic not just to some *graph template* that is represented in *sc-memory*, but to the *program object* of this *graph template*, i.e. *graph template*, which is presented in a program format convenient for quick processing.

∋ *Method for finding sc-memory constructions isomorphic to the specified graph template*
∈ *method*
⇒ *method header in method representation language*:*
[ScTemplate::Result HelperSearchTemplate(ScTemplate const & templ, ScTemplateSearchResult & result)]
∈ *C++*
⇒ *input argument classes of a method*:*
⟨
- *graph template program object*
- *tuple of program objects of sc-memory constructions isomorphic to the specified graph template*

⟩
⇒ *method result class*:*
- *error code of the result of creating a program object by the specified element corresponding to the graph template*

⇒ *class of exceptions*:*
- *syntactically incorrect graph template*