PROCEDURE FOR DEBUGGING REPLICA CLASS ERRORS IN RULE-ORIENTED KNOWLEDGE BASES

RESEARCH GROUP:

Serhii Osieievskyi
Candidate of Technical Sciences, Associate Professor, Lead Researcher of Air Force Scientific Center
Ivan Kozhedub Kharkiv National Air Forces University, Ukraine

Oleksii Kolomiitsev
Honored Inventor of Ukraine, Doctor of Technical Sciences, Senior Researcher, Professor at the Department
National Technical University “Kharkiv Polytechnic Institute”, Ukraine

Viacheslav Tretiak
Candidate of Technical Sciences, Associate Professor, Lead Researcher of Air Force Scientific Center
Ivan Kozhedub Kharkiv National Air Forces University, Ukraine

Oleksii Petrenko
Candidate of Technical Sciences, Senior Researcher, Deputy Cief of Scientific Department of Air Force Scientific Center
Ivan Kozhedub Kharkiv National Air Forces University, Ukraine

Petrenko Olha
Candidate of Technical Sciences, Associate professor, Associate professor, Dep. of Information Technologies Security
National University of Radio Electronics, Ukraine

Andrii Vlasov
Candidate of Technical Science, Senior Researcher, Associate professor of the department of Information technology security
Kharkiv National University of Radioelectronics, Ukraine

Ruslan Oliinyk
Chief of Section
State Scientific Research Institute of Armament and Military Equipment Testing and Certification, Chernihiv, Ukraine
Yurij Zhivetc
Researcher
State Scientific Research Institute of Armament and Military Equipment Testing and Certification, Chernihiv, Ukraine

Oleksandr Shumigay
Researcher
State Scientific Research Institute of Armament and Military Equipment Testing and Certification, Chernihiv, Ukraine

Pavlo Baturin
Cadet
Ivan Kozhedub Kharkiv National Air Forces University, Ukraine

Summary. The problems of error detection by experts in the rule-oriented knowledge bases of oriented information systems, which occurring at the stage of the knowledge base formation, are considered. It has been determined that such errors are associated with the contradictory opinions of experts and/or the limited/imperfect description of the subject area. The approaches to their debugging were analyzed. The ways of improving the existing approaches to debugging static errors of the “replica” class and possible ways of applying the obtained solutions to eliminate errors “contradictoriness”, “redundancy”, “incompleteness” have been shown. Considering the requirements for the accuracy of information presentation, the solutions for expanding the content of the definition of the static error of the rule-oriented knowledge base have been proposed. The issues of the influence of the “replica” class errors on the results of output in accordance with the rules of the rule-oriented knowledge base were considered. The possibility of using graph theory methods for solving the problem of debugging errors of the “replica” class has been proved. The algorithmic structures for identifying and debugging errors of the specified class, which in contrast to existing solutions allow to identify duplicate apexes at each rank of the graph to which the rule-oriented knowledge base is amounted to, have been developed. A software implementation of detecting and debugging static errors of incomplete, fragmentary and complete duplication has been developed. Due to usage of recursion, the requirements for preparing an array of input data for processing were reduced. The solutions obtained are compatible with the requirements of DSTU ISO / IEC 9126, DSTU ISO / IEC 14598 and consider the requirements of the Software Quality Requirements and Evaluation family of standards as the value of the apexes of the event tree graph. In the process of solving the problem, the specifics of the functioning of the man-machine system was considered. In particular, the possibility of formalizing various aspects of knowledge (alethic, dissisional, casual, dianistic) and assurance of a given level of operativeness in finding solutions has been accounted.

Key words: graph, model, technique, rule-oriented model, knowledge base.

1. Introduction

Comprehensive increasing in the capabilities to decision-making automation in poorly formalized areas through the usage of expert’s knowledge result in an increase in the number of errors in the software of knowledge-based information systems (KBIS). Based on a detailed analysis it was concluded in [1] that the problem of error detection and elimination becomes sharper as the complexity of the solved by the software tasks, and threatens catastrophes in systems that perform critical management functions of large, expensive and especially important objects or
processes. These aspects require the development of appropriate algorithms, models and methods that will ensure the required quality of software in general and KBIS software, in particular. One of the main levers for achievement of high reliability of proper functioning and quality of KBIS software is diagnosing software at different stages of its life cycle, in particular, at the stages of design, development and operation. The important components of diagnosing KBIS software are debugging and testing. Their role increases with the putting in the assumption that the KBIS software is quite complex and cannot be unfailing. For this reason, identifying the limitations of the methods and means of debugging and testing of KBIS software, including the identification of hidden errors of applications, is an urgent task of developing and improving the system of support for KBIS software development processes.

2. Analysis of research and publications

The possible ways to solve the problem of developing methods for debugging knowledge bases (KB) are described in detail in the works of such scientists as V. Lipaev, S. Zikova, A. Narinyani, L. Zadeh, Y. Tsukamoto, S. Miller [2-6] etc. From the analysis of these works and publications it is clear that the greatest success has been achieved in the development of methods of static debugging, i.e. those methods that do not require an expert system staring up for implementation. However, at the moment there is no single approach to the formalization of so-called structural errors, such as redundancy, incompleteness, contradictoriness, which can be recognized by methods of static debugging. That is why statically correct knowledge bases do not guarantee the quality of decisions due to errors in the knowledge itself, which often associated with the complexity of a particular subject area.

3. Research objective

Considering the results of the analysis and available solutions in the field of KBIS design, the objective of this publication is determined as mechanisms developing for identifying duplicates of the knowledge bases (KB) model to support the development of intelligent decision support system, which will be able to eliminate these problems.

4. Research outcomes

4.1 General approach to the development of a methodology for debugging knowledge bases

According to [8], debugging is a methodical process of finding and reducing the number of errors or defects in a computer program or electronic equipment in order to obtain the expected behavior. Debugging is usually becoming more complicated when different subsystems are strongly connected, because changes in one part can cause errors in another. Put in other words the aim of the debugging knowledge bases process is to identify and eliminate the maximum number of errors in the KBIS software.

Errors of the knowledge base are generally regarded as divided into the following groups:
- errors related to the internal structural imperfection of the knowledge base;
- errors related to external contradictions of the knowledge base.

To identify the first group of errors, the knowledge base is presented in the form of a graph model and formalize all possible errors associated with the internal
structural imperfection of the knowledge base. Such errors include incompleteness, redundancy, internal contradictoriness. The next step is to identify errors in this group and correct the knowledge base by bringing it to a state of static correctness. By virtue of the fact that the term static correctness most fully reflects the essence of solving the problem, next in the work it will be used by considering its interpretation in [8, 9].

It is important to note that establishing the fact of static correctness of the knowledge base does not guarantee that the statically correct knowledge base does not draw incorrect conclusions due to possible errors in the rules themselves. The causes of these possible errors are external contradictoriness of the subject area itself and errors in the rules themselves. Next the definition in outward appearance contradictory knowledge base in the narrow sense of the word can be introduced.

It should be noted that most subject areas by their nature are outwardly contradictory and contain errors related to limitations in the subject area or the presence facts of the critical combination of events that lead to errors in decision-making. Obviously that the indicated above errors are not related to the structure of knowledge and cannot be detected by methods of graphs and methods of static analysis. The conclusion follows from this that the task of detecting such errors can be realized only by testing the knowledge base itself. And, as a result, there is a statement that static debugging, during which structural errors of the knowledge base and internal contradictoriness are detected and eliminated, is a necessary but not sufficient condition for ensuring the quality of the KB.

As mentioned above, to identify errors associated with the internal structural imperfection of the knowledge base, it must be corrected to a graph model. In our case (so far as a static analysis of the knowledge base is concerned) it is assumed that the structure of the KB is known. This fact allows to build a graph model (of structure AND / OR) and perform its further analysis. It should be entered a restriction at this stage, which is as follows. These solutions are an effective mechanism for identifying errors associated with the internal structural imperfection and external contradictions of the knowledge base only for rule-oriented knowledge bases. This is due to the fact that for the neural network, the structure of knowledge which was obtained by the neural network during training is implicit; it is formed in a set of weight coefficients of connections between neurons which cannot be interpreted.

In addition, the most balanced structure of the neural network does not allow to say for sure that the presence of such classes of errors as “incompleteness”, “redundancy” and “contradiction”.

4.2 Models of errors in rule-oriented knowledge bases

For rule-oriented knowledge bases (ROKB) KBIS in the literature [10 - 22] the following main classes of errors were distinguished: incompleteness, redundancy, contradiction. However, despite the available papers in this branch, there is still no systematic approach to formalize the main types of structural errors. In this paper, it is proposed to make in a separate class “Replicas” errors such as “Duplicates” and consider them in particular. The main classes of ROKB structural errors are presented in Figure 1. The class “Replicas” is separated out, and proposed for consideration.
Structural errors of the rule-oriented knowledge base will have been formalizing in terms of graph theory. In order to visualize the immediate theoretical positions, as a structure of the rule-oriented knowledge base under consideration, we will use a hybrid network model of knowledge about the process of selecting sources of orders of fire means of the Air Force, given in [17] and considering the content of [18].

![Diagram of structural errors ROKB]

Fig. 1. **Classes of structural errors of the rule-oriented knowledge base**

The formalized description of the fragment of the above model in terms of ROKB will be as follows:

\[ r_1: \text{if } s_1 \text{ and } s_2, \text{ then } f_1; \]
\[ r_2: \text{if } s_2 \text{ and } s_3, \text{ then } f_2; \]
\[ r_3: \text{if } s_3 \text{ and } s_4, \text{ then } f_3; \]
\[ r_4: \text{if } f_1, \text{ then } g_1; \]
\[ r_5: \text{if } f_2 \text{ and } f_3, \text{ then } g_2; \]

where \( s_1, s_2, s_3, s_4 \in S, F; \)
\( f_1, f_2, f_3 \in F; \)
\( r_1, r_2, r_3, r_4, r_5 \in R; \)
\( g_1, g_2 \in G. \)

where \( F \) is the set of facts of the KBIS, which combines input, intermediate and terminal facts; \( R \) is a set of rules that have the form \( R: \text{if } (f_{i_1} \& f_{i_2} \& \ldots \& f_{i_k}) \text{ then } f_{i(i+1)}. \)
The connections formed in the structure of the graph will be called output chains. Put in other words, in general, the output chain $l_i$ is a sequence of rules $(r_{l_1}, r_{l_2}, r_{l_3}, \ldots, r_{l_k})$ such that $\forall r_{l_i}, r_{l_{i+1}}, q \in D_{r_{l_{i+1}}}$ on condition that $k = 2, \ldots, n - 1$, where $D_r$ is the set of facts belonging to the conditional part of the rule $r$. Then for the graph shown in Figure 2, $L$ is the output chain, where $L = \{l_1, l_2, l_3, l_4, l_5, l_6, l_7, l_8\}$, where $l_1 = (r_1), l_2 = (r_2), l_3 = (r_3), l_4 = (r_4), l_5 = (r_5), l_6 = (r_1, r_4), l_7 = (r_2, r_5), l_8 = (r_3, r_5)$. It is logical that for the output chain $l_i$ the beginning of the output chain $l_i$ of the form $(r_{l_1}, r_{l_2}, r_{l_3}, \ldots, r_{l_k})$ is a set of facts in the condition of the first rule, $D_{l_i} = D_{r_{l_1}}$, and the end of the output chain $l_i$ is the result of the last rule, $G_{l_i} = G_{r_{l_i}}$.

The foregoing description reflects the general approach to the formalization of structural errors of the rule-oriented knowledge base in terms of graph theory and will be used in the further presentation of the material.

### 4.3 Formalization of structural errors of the “Replica” class

According to the classification of structural errors, the “Replica” class contains three components: incomplete duplicates, fragmentary duplicates and complete duplicates. We formalize each component of this class will be formalized and the algorithms for their detection will be given.

Rules $r_i$ and $r_j$ are called duplicates if they have at least one common fact in execution conditions and the same result of execution. I.e. formally the task of finding incomplete duplicates for the graph $G_s$ is that for a given graph KBIS system $G_p$ it is necessary to find pairs of vertices from subset $V_s$ (subset of vertices), connected by two (or more) oriented chains of rank 2. Figure 3 shows an example of incomplete duplicates.
The formalized description of the graph $G_s$ shown in Figure 3 is as follows:

- $r_1$: if $f_1$ and $f_2$, then $f_6$
- $r_2$: if $f_2$ and $f_3$, then $f_6$
- $r_3$: if $f_4$ and $f_5$, then $f_7$

$D_{l_1} \cap D_{l_2} = f_2$

Measures to correct the “incomplete duplicates” error for each case are determined by the expert and depend on the depth of duplicate nesting in the general system of ROKB rules.

The task of finding incomplete duplicates is solved by using a modified bypass by width procedure for a given graph and initial vertices from the set $V_1$. The search in width is not carried out until all reachable vertices of the graph are exhausted, but only to the vertices that are in the second rank from the initial one. The structure of the modified algorithm of the procedure for finding incomplete duplicates for the graph $G_s$, which determines the error of the type “incomplete duplicates” for any vertex $f$ is presented in Figure 4.

![Diagram](image-url)

**Fig. 4.** Structure of the modified algorithm of the procedure for finding incomplete duplicates for the graph $G_s$
A brief explanation of the algorithmic structure will be given. At the preparation stage, an array of vertices for which there is a bidirectional connection with the target vertex is formed and their logging to the queue is performed.

For all vertices in the queue (which rank is 1), the operation of removing from the queue and remembering the current position of the two-way communication is performed. Next, revision of the adjacent vertices is performed. If an attempt to re-add a vertex to the queue occurs (the vertex is not new and has already been revised for an existing rule) the vertex is marked as a duplicate and its rank is fixed.

```java
import java.util.*;
public class Vertex {
    private final int level;
    private final String name;
    private final List relations = new ArrayList();
    public Vertex(int level, String name) {
        this.level = level;
        this.name = name;
    }
    public void addRelation(Vertex child) {
        relations.add(child);
    }
    public List<Vertex> getChildren() {
        return relations;
    }
    public static HashMap<Integer, List<Vertex>> findDuplicates(List<Vertex> graph) {
        List<Vertex> nextLevelVertexes = new ArrayList();
        for (Vertex vertex : graph) { // searching for the all high rank vertexes
            nextLevelVertexes.addAll(vertex.getChildren());
        }
        Set<Vertex> listOfCheckedVertexesVisuotDuplicates = new HashSet<>(); // unique vertexes
        HashMap<Integer, List<Vertex>> duplicatesMap = new HashMap<>(); // integer - vertex level, the list of duplicates
        if (!duplicatesMap.containsKey(vertex.level)) {
            duplicatesMap.put(vertex.level, new ArrayList<Vertex>());
            duplicatesMap.get(vertex.level).add(vertex);
        } else {
            // if not a duplicate – save to an array of unique
            listOfCheckedVertexesVisuotDuplicates.add(vertex);
            // if there are lower levels than forming their list for recursive calculation of duplicates is performed
            if (vertex.getChildren() != null && !vertex.getChildren().isEmpty()) {
                childOfChildList.add(vertex);
            }
        }
    }
    @Override
    public Boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Vertex vertex = (Vertex) o;
        return level == vertex.level && Objects.equals(name, vertex.name) &&
            Objects.equals(relations, vertex.relations);
    }
    @Override
    public int hashCode() {
        return Objects.hash(level, name, relations);
    }
}
```

**Fig. 5.** Fragment of the listing of the procedure for finding incomplete duplicates by all ranks of the graph
The complexity of the duplicate search algorithm for one fact does not superior to the complexity of bypassing the width in the worst case $O(V + E)$ [19]. But by virtue of the fact that algorithm is repeated for each rank of KBIS, then, the complexity of finding incomplete duplicates of the population $O((V + E) \times Level)$, where $Level$ is the number of ranks of KBIS. Figure 5 shows a fragment of the listing of the procedure for finding incomplete duplicates at all ranks of the graph $G_s$.

This procedure for finding incomplete duplicates specifies working with unordered lists, which significantly reduces the time of execution of data array preparation operations.

The presence of fragmental duplicates in the KBIS is determined by the situation when the rules $r_i$ and $r_j$ are relevant to $D_{r_i} \subset D_{r_j}$, while $D_{r_i} \neq D_{r_j}$ and, respectively, $q_{r_i} \subset q_{r_j}$. In other words, the set of facts that belongs to the conditional part of the rule $r_i$ is part of the set of facts that belongs to the conditional part of the rule $r_j$. For instance, rules $r_1$ and $r_2$ in Figure 6 are fragmentary duplicates.

The formalized description of the graph $G_s$ is shown in Figure 6 as follows:

$$r_1: \text{if } f_1 \text{ and } f_2, \text{ then } f_4;$$
$$r_1: \text{if } f_3 \text{ and } f_2 \Rightarrow f_1, \text{ then } f_4;$$
$$D_{r_1} = \{f_1, f_2\};$$
$$D_{r_2} = \{f_1, f_2, f_3\};$$
$$D_{r_1} \subset D_{r_2};$$
$$q_{r_1} = f_4; q_{r_2} = f_4.$$

![Fig 6. Example of fragmentary duplicates (graph $G_s$)](image)

The presence of a fragmentary duplicate in the formal description indicates that in the graph $G_s$ there are two vertices $r_1$ and $r_2$ such that the following conditions are carried out simultaneously:

- $\exists f \in V_1(r_1, f) \in E \& V_1(r_1, f) \in E$ (the fact of belonging to part of the output rule);
- $\forall f \in V_1(f, r_1) \in E \Rightarrow V_1(f, r_2) \in E$ (inclusion of part of the condition of the output rule $r_1$ into the part of condition of the output rule $r_2$);
- $\exists f \in V_1(r_1, f) \in E \& V_1(r_2, f) \notin E$ (strict inclusion of parts of the condition of the rules $r_1$ and $r_2$).

The algorithmic solution of this problem is identical to the solution of the problem of finding incomplete duplicates with the condition of changing the ranks.
of the target vertices to odd ones. Assessment of complexity of algorithm of partial duplicates search depends on characteristics of data structures representing the graph \( G \). And the only thing that can be affected by developer is to use the union-find solution to save data structures, because the application of such a solution requires to represent the entire structure of the graph \( G \); the memory size, which is proportional to the number of arcs of the specified graph.

The task of finding complete duplicates for the graph \( G \) lies in the fact that for a given graph of the expert system \( G_p \) it is necessary to find pairs of vertices from the subset \( V_1 \), such that the set of their output vertices is unique and the set of input vertices satisfies the strict inclusion ratio. Rules \( r_1 \) and \( r_2 \) in Figure 7 are complete duplicates.

![Fig 7. Example of complete duplicates (graph \( G_s \))](image)

The formalized description of the graph \( G_s \) is shown in Figure 7 as follows:

\[
\begin{align*}
  r_1 &: \text{if } f_1 \text{ and } f_2 \text{ and } f_3 \text{ and } f_4, \text{ then } f_5; \\
  r_2 &: \text{if } f_1 \text{ and } f_2 \text{ and } f_3 \text{ and } f_4, \text{ then } f_6; \\
  r_3 &: \text{if } f_3 \text{ and } f_4, \text{ to } f_6.
\end{align*}
\]

The solution to the problem of debugging complete duplicates is seen in the removal of all duplicates except one. The presence of complete duplicates indicates the absolute coincidence of two or more rules in ROKB. From the formalized description follows the presence of matching parts of the conditions for the rules of inference \( \forall f \in V_1(f, r_1) \in E \& V_1(f, r_2) \in E \& V_1(f, r_3) \in E \).

The algorithmic structure for searching for complete duplicates is identical to the algorithmic structure of the procedure for searching for incomplete duplicates and differs only in the choice of both even and odd ranks of arrangement of target vertices.

It should be noted that the developed algorithmic structure and procedure for finding duplicates can be modified by preliminary analysis of the nature of the error. This will allow to reduce the time to determine the rank of the target vertices.

5. Conclusion

Thus, along the research, it has been established that in the rule-oriented knowledge bases of KBIS it is possible the presence of the following classes of errors that occur at the stage of formation of the knowledge base by experts:
- internal contradictoriness, redundancy, incompleteness, replica;
- internal errors of knowledge base rules;
- errors related to the external contradictoriness of the subject area.

Conclusions on their impact on the quality of KBIS software in general have been formed for the “Replica” class. On the basis of the researching of the nature of this class of errors a mechanism for bringing the logical structure of the rule-oriented knowledge base to the graph form has been developed. It is shown that this error class is structural and can be detected by static debugging based on graph methods. Algorithmic structures for finding errors of the “Replica” class have been developed and their computational complexity has been determined. Mentioned algorithms are implemented in the form of software procedures and can be included into the body of information component of KBIS developing process support. Researches, that have been done, have shown that debugging of the "Replica" class errors transform the rule-oriented knowledge base to a state of static correctness, which is a necessary condition for debugging. Further research may be aimed at developing procedures for derecognition of error classes “redundancy” and “incompleteness”.

References:


