# Conceptual Knowledge Extraction and Retrieval Within Decomposition of Classes and Objects

Dmytro O. Terletskyi[1], Sergey V. Yershov[2]

[1] *V. M. Glushkov Institute of Cybernetics of NAS of Ukraine, Academician Glushkov Avenue, 40, Kyiv, 03187, Ukraine*
[2] *V. M. Glushkov Institute of Cybernetics of NAS of Ukraine, Academician Glushkov Avenue, 40, Kyiv, 03187, Ukraine*

**Abstract**

An ability to extract hidden and implicit knowledge, their integration into a knowledge base, and then retrieval of required knowledge items are important features of knowledge processing for many modern knowledge-based systems. However, the complexity of these tasks depends on the size of knowledge sources, which were used for extraction, the size of a knowledge base, which is used for the integration of extracted knowledge, as well as the size of a search space, which is used for the retrieval of required knowledge items. Therefore, in this paper, we adapted the algorithm for the decomposition of homogeneous classes of objects, within such knowledge representation model as object-oriented dynamic networks, to perform dynamic knowledge extraction and retrieval, adding additional filtration parameters. As the result, the algorithm extracts knowledge via constructing only semantically consistent subclasses of homogeneous classes of objects and then filters them according to the attribute and dependency queries, retrieving knowledge. To demonstrate some possible application scenarios for the improved algorithm, we provided an appropriate example of knowledge extraction and retrieval via decomposition of a particular homogeneous class of objects. In addition, we developed algorithms for objects' decomposition for their run-time usage and database storage, which allow the generation of instances for subclasses created during the class decomposition, and to perform knowledge extraction on the object level.

**Keywords**

Internal semantic dependencies, decomposition consistency, decomposition of classes, knowledge extraction, knowledge retrieval.

## 1. Introduction

The extraction and retrieval of knowledge are important features of many modern knowledge-based systems. Such systems are capable to extract new knowledge by analyzing relevant knowledge sources, integrating it with previously obtained knowledge, and allowing users to search for necessary knowledge items in the knowledge base. Depending on the chosen knowledge representation model, the extraction of new implicit and hidden knowledge can be implemented in different ways. For object-oriented knowledge representation models, knowledge extraction can be performed via universal exploiters of classes, such as union, intersection, difference, and decomposition, which allow the construction of new classes of objects based on the existed ones.

In this paper, we study the decomposition of homogeneous classes of objects, within such knowledge representation model as object-oriented dynamic networks, to demonstrate that the algorithm for decomposition of classes can be used as a tool for knowledge extraction and retrieval. For this purpose, we improved the algorithm for decomposition of homogeneous classes of objects, which was proposed in [28], by adding more additional parameters, that allow adaptation of the algorithm, developed for knowledge extraction, to dynamic knowledge retrieval. We also discovered that classical

[1]13th International Scientific and Practical Conference on Programming UkrPROG'2022, October 11–12, 2022, Kyiv, Ukraine
EMAIL: dmytro.terletskyi@nas.gov.ua (A. 1); ErshovSV@nas.gov.ua (A. 2)
ORCID: 0000-0002-7393-1426 (A. 1); 0000-0002-9895-777X (A. 2)

methods of formal concept analysis do not cover internal semantic dependencies among properties and methods of homogeneous classes of objects. In addition, we show how the improved algorithm can reduce the search space during the retrieval of implicit or hidden knowledge, which cannot be obtained using standard methods of formal concept analysis.

## 2. Formal Concepts Analysis

Among the variety of formal systems for the analysis and processing of conceptual knowledge, formal concept analysis is one of the most developed frameworks, which is based on the mathematical theory of lattices. It provides tools for the construction, analysis, and processing of conceptual hierarchies, represented in terms of two isomorphic complete lattices of objects and attributes. Since lattices consist of chains, which are posets, it allows inference and retrieval of new concepts within the corresponding formal context. Let us consider the main concept of the formal concept analysis described in [20, 21]. The first step is the definition of the formal context.

**Definition 1.** *A formal context is a tuple* $(G, M, I)$*, where* $G$ *is a set of objects of the context, while* $M$ *is a set of its attributes, and* $I$ *is a relation between* $G$ *and* $M$*, which express that an object* $g \in G$ *has an attribute* $m \in M$*, i.e.* $(g, m) \in I$ *or* $gIm$*.*

Using this definition, any formal context can be represented by a corresponding cross table, where columns mean attributes, while row mean objects. It allows considering a set of common attributes for a set of objects, and a set of objects that have attributes from a set of common attributes.

**Definition 2.** *A set of common attributes for selected set of* $A \subseteq G$ *objects* $A^{\uparrow} = \{m \in M \mid gIm \ \forall g \in A\}$ *is a set , i.e. all attributes from the set* $A^{\uparrow}$ *are common for all objects from the set* $A$*.*

**Definition 3.** *A set of objects with the common attributes* $B \subseteq M$ *is a set* $B^{\downarrow} = \{g \in G \mid gIm \ \ \forall m \in B\}$*, i.e. all objects from the set* $B^{\downarrow}$ *have all attributes from the set* $B$*.*

Using these notions, we can define a formal concept based on a particular formal context

**Definition 4.** *A formal concept of the formal context* $(G, M, I)$ *is a pair* $(A, B)$*, where* $A \subseteq G$ *is an extent of the formal concept, while* $B \subseteq M$ *is an its intent, and where* $A^{\uparrow} = B$*,* $B^{\downarrow} = A$*.*

Such definition of the formal concepts, i.e. using the notions of an extent and an intent, is similar to combination of two ways of set definition, described in [22]. In the first case, a set can be defined by particular elements (tabular form), while in the second one, it can be determined using the attributes, which must have all elements of the set (set builder from). In addition, according to [16], the notion of a formal concept is also similar to a combination of two theoretical forms of class consideration – an extensional and an intensional. From the first perspective, a class can be defined by the list of its objects, while from the second one it can be defined by the set of attributes. The definition of the formal concept proposed in [20] combines these two perspectives into a single notion and provides an opportunity simultaneously to consider a particular formal concept using both of them.

Since a formal context can define a certain number of formal concepts, there is a sense to define a set of all formal concepts.

**Definition 5.** *A set of all formal concepts of the formal context* $(G, M, I)$ *is a set* $PS(G, M, I)$*.*

Formal concept analysis has different applications within an area of knowledge processing. According to [31], *conceptual knowledge retrieval* is one of the main categories among the variety of methods of formal knowledge processing. On another side, these methods allow the implementation of corresponding functionality within knowledge-based systems developed based on formal concept analysis. In general, the knowledge retrieval task can be simply described as querying a knowledge base to find the required knowledge items. According to [7, 9-11, 15, 17, 18, 31], the formal concept analysis allows defining a formal context, where the intent of the context is defined by pieces of

knowledge, for example, keywords or part of sentences, while the extent is defined by the list of documents, that contain or do not contain such knowledge items. The corresponding concept lattice, constructed based on the formal context, describes the search space, consequently, the retrieval process can be interpreted as the matching of the search query with the formal concepts, which are represented by lattice nodes, using different search strategies based on the relations of generalization and specialization defined between formal concepts. The performance of the retrieval process depends on the size of the search space and the corresponding search strategy. Therefore, as was noted in [32], one of the main goals for many retrieval algorithms is to reduce the search space as much as possible. Another issue related to query matching is the correspondence level of each formal concept to the query, as it can be rather partial than complete. Thus, in the many search strategies queries are described in a form of inclusion conditions, which allow the handling of partial query matching within the concept lattice.

## 3. Morphology of Classes

Nowadays, there are a few approaches, which propose the application of the formal concept analysis to studying dependencies within the procedural program constructions. One of such was proposed in [30], the main idea of which is to construct the concept lattice of decomposition slices of the program. It provides an opportunity to analyze groups of the ordered program statements, called decomposition slices, related to a particular context, for example to a variable. In other words, each particular variable, which is a part of the program, depends on the corresponding ordered sequence of operators, which somehow use or change its state. However, the proposed approach was designed for procedural programs, but not for object-oriented ones. Consequently, it is more suitable for the structure analysis of procedural knowledge than for the analysis of declarative knowledge represented in terms of classes. A similar approach, but for the analysis of class methods cohesion, was proposed in [29]. The main idea is to consider dependencies between different program statements within a particular method of a class and define the corresponding formal context using them, and then construct the concept lattice called a cohesion lattice. However, the approach does not pay the attention to the external dependencies of class attributes used in the method with other properties and methods of the class, which are important for the decomposition of homogeneous classes of objects.

Usually, a formal context is defined using a set of attributes and a set of objects, where attributes have corresponding values encapsulated in a particular object. However, a formal context can also be determined using a set of classes and a set of attributes. This idea was used in [13, 14] to analyze the structure of classes in the Java programming language, in particular, to consider the interrelation between methods call of a class and then optimize its structure. The embedded call graph provides additional information about the interaction between methods of the class, which is absent in the corresponding concept lattice. However, such an approach covers only dependencies between methods of the class and does not pay the attention to other kinds of dependencies, for example, between properties, properties, and methods of the class. Another application of the class formal context was proposed in [23], which was used to analyze the class cohesion via the construction of the corresponding concept lattice called cohesion lattice. They capture the cohesiveness of a class and its members, which provides an opportunity to reorganize the class structure more efficiently, increasing cohesion. However, cohesion metrics are rather quantitative measures of dependencies among class members, than qualitative. Many approaches to class cohesion measurements pay the attention to dependencies' existence but not to their semantics and consistency within a modeled domain, which is crucial for the decomposition of homogeneous classes.

One of the known approaches, that considers dependencies between the attributes of an object, is the detection of functional dependencies in relational databases. As it was noted in [4-6, 8, 32], a functional dependency is defined as the implication over the relation pairs, determined on the set of attributes, which are mapped into columns of particular tables. The main idea of functional dependencies is to conclude that if two particular tuples of attributes in the relation contain a certain attribute $X$, which is called an antecedent, then they also contain another attribute $Y$, which is called a consequent. Such facts can be considered new knowledge, which is hidden or implicit. In addition, there is a generalized form of functional dependency called a similarity dependency [4-6, 8], the main idea of which is the satisfaction of functional dependency for any two tuples in the relation. However, such

kinds of dependencies do not cover the internal semantic connection within classes and objects because they consider only the availability of a particular attribute for an object, rather how the different attributes of the object are related to each other, or more precisely, how they depend on each other. They do not consider how the presence or absence of one particular attribute for an object or a class affects their semantic consistency.

An alternative approach to the analysis of dependencies between attributes of objects was proposed in [25], according to which an attribute $m_2$ depends on another attribute $m_1$, i.e. $m_1 \succ m_2$, whenever the presence of $m_2$ is not significant without the presence of $m_1$, where $m_1$ and $m_2$ also may be atomic, as well as conjunctive or disjunctive attributes. However, such an explanation of the dependency between attributes is quite fuzzy because it is unclear how to verify that presence of one attribute is not significant without the presence of another one, as well as what the term significance should be meant here.

To consider what internal semantic dependencies of a class are, their kinds, and how they affect the decomposition of the homogeneous classes of objects, let us consider the definitions of a homogeneous class of objects and its subclass within such knowledge representation model as object-oriented dynamic networks (OODNs), which was proposed in [26, 27].

**Definition 6.** *The homogeneous class of objects* $T$ *is a tuple* $T = (P(T), F(T))$, *where* $P(T) = (p_1(T), ..., p_n(T))$ *is a collection of properties which define the structure of the class* $T$, *while* $F(T) = (f_1(T), ..., f_m(T))$ *is a collection of its methods, that define its behavior.*

**Definition 7.** *A homogeneous class of objects* $T_i$ *is a subclass of homogeneous class of objects* $T$, *i.e.* $T_i \subseteq T$, *if and only if* $P(T_i) \subseteq P(T)$ *and* $F(T_i) \subseteq F(T)$, *where* $P(T_i)$, $P(T)$ *and* $F(T_i)$, $F(T)$ *are specifications and signatures of the class* $T_i$ *and* $T$, *respectively.*

Let us consider an example of a homogeneous class of objects and analyze its specification and signature to understand how the properties and methods can depend on each other, creating internal semantic dependencies. For this purpose, let us define a homogeneous class of objects $Pt$, which describes a concept of a point on a plane, and has the following structure:

$$\begin{aligned} Pt(&p_1 = (x, (v_x, \mathbb{R})),\\ &p_2 = (y, (v_y, \mathbb{R})),\\ &f_1 = get\_x(pt, \mathbb{R}),\\ &f_2 = get\_y(pt, \mathbb{R}),\\ &f_3 = set\_x(pt, (x, \mathbb{R})),\\ &f_4 = set\_y(pt, (y, \mathbb{R}))), \end{aligned}$$

where $Pt.p_1$ and $Pt.p_2$ are quantitative properties, which mean coordinates $(x, y)$ of a point $pt$; $Pt.f_1$ and $Pt.f_2$ are methods, which return $x$ and $y$ coordinates of a point $pt$, respectively; $Pt.f_3$ and $Pt.f_4$ are methods, which provide an opportunity to set a value of $x$ and $y$ coordinates of a point $pt$, respectively.

Now let us define another homogeneous class of objects $Tr$, which describes a concept of a triangle on a plane, and has the following structure:

$$
\begin{aligned}
Tr(\ & p_1 = (vertex_1, (v, Pt)),\\
& p_2 = (vertex_2, (v, Pt)),\\
& p_3 = (vertex_3, (v, Pt)),\\
& p_4 = is\_a\_triangle(vf_4(tr), v \in \{0,1\}),\\
& f_1 = get\_vertex(tr, (i, \mathbb{Z}^+), Pt),\\
& f_2 = set\_vertex(tr, (i, \mathbb{Z}^+), (a_x, \mathbb{R}), (a_y, \mathbb{R})),\\
& f_3 = compute\_side\_length(tr, (vertex_a, Pt), (vertex_b, Pt), \mathbb{R}^+),\\
& f_4 = compute\_perimeter(tr, \mathbb{R}^+)),
\end{aligned}
$$

where $Tr.p_1$, $Tr.p_2$, and $Tr.p_3$ are quantitative properties, which mean vertices of a triangle, defined as objects of the class $Pt$; $Tr.p_4$ is a qualitative property, which means satisfiability of the triangle inequality and is defined by the following verification function:

$$
\begin{aligned}
& vf_4(tr) : (tr.vertex_1, tr.vertex_2, tr.vertex_3) \to \{0,1\},\\
& vf_4 = ((s_1 + s_2 > s_3) \wedge (s_1 + s_3 > s_2) \wedge (s_2 + s_3 > s_1)),
\end{aligned}
$$

where $s_1$, $s_2$, and $s_3$ are defined as follows:

$$
\begin{aligned}
& s_1 = compute\_side\_length(tr.get\_vertex(1),\ tr.get\_vertex(2)),\\
& s_2 = compute\_side\_length(tr.get\_vertex(1),\ tr.get\_vertex(3)),\\
& s_3 = compute\_side\_length(tr.get\_vertex(2),\ tr.get\_vertex(3)),
\end{aligned}
$$

$Tr.f_1$ is a method, which returns the coordinates $(x, y)$ for a vertex $i$ of a triangle $tr$ in a form of objects of the class $Pt$ and is defined as follows $f_1(tr, i) = (tr.vertex_i)$; $Tr.f_2$ is a method, that set coordinates $(a_x, a_y)$ for a vertex $i$ of a triangle $tr$ and is defined as follows

$$
f_2(tr, i, a_x, a_y) = (tr.vertex_i.set\_x(a_x), tr.vertex_i.set\_y(a_y));
$$

$Tr.f_3$ is a method, which returns a distance between $vertex_a$ and $vertex_b$ of a triangle $tr$, and defined as follows:

$$
\begin{aligned}
& f_3(tr, vertex_a, vertex_b) = \sqrt{d_1 + d_2},\\
& d_1 = (vertex_a.get\_x() - vertex_b.get\_x())^2,\\
& d_2 = (vertex_a.get\_y() - vertex_b.get\_y())^2,
\end{aligned}
$$

$Tr.f_4$ is a method, which returns the perimeter of a triangle $tr$ and is defined as follows $f_4(tr) = s_1 + s_2 + s_3$, where $s_1$, $s_2$, and $s_3$ are defined in the same way as in for the $vf_4(tr)$.

Let us consider the class $Tr$ as a collection of properties and methods, i.e.

$$
Tr = P(Tr) \cup F(Tr) = \{Tr.p_1, Tr.p_2, Tr.p_3, Tr.p_4, Tr.f_1, Tr.f_2, Tr.f_3, Tr.f_4\}.
$$

We denote an $i$-th property of the class $Tr$ by $Tr.p_i$, and a $j$-th method – by $Tr.f_j$ for a more compact representation of all statements noted below.

As it was noted in [32], for the class $Tr$ we can construct $2^n = 2^8 = 256$ subclasses, which create the power set lattice $L = (PS(Tr), \subseteq, \cup, \cap)$, which is a complete lattice and where $PS(Tr)$ is a set of all possible unique subsets of the set $P(Tr) \cup F(Tr)$. From the decomposition perspective, we

need to consider only $2^n - 1$ subclass, which are nonempty ones and create the join-semilattice $JSL = (PS(Tr) \setminus \{\varnothing\}, \subseteq, \cup)$, which describes the search space for the knowledge retrieval. This semilattice is not a concept lattice, however as it was demonstrated in [28], it can crucially reduce the space search for the solving decomposition constraint satisfaction problem.

To compare the opportunities provided by the join-semilattice of nonempty subclasses of the homogeneous class of objects $Tr$ and the concept lattice of all its subclasses, let us define the formal context of all possible subclasses of the class $Tr$ and then construct the corresponding concept lattice. For this purpose, let us consider the formal context $W_1 = (G = PS(Tr),\ M = P(Tr) \cup F(Tr),\ I : G \times M)$ and define the corresponding cross table for it. We do not provide a full cross table here because of its size, however, Table 1 illustrates the basic intuition for the definition of formal context $W_1$. We use the symbol plus $+$ to specify the pair of the relation $(g, m) \in I$. An object $g$ is defined as follows $g \in SC_j^i(Tr) \subseteq Tr$, $i = \overline{0, |M|}$, $j = \overline{1, C_{|M|}^i}$, where $SC_j^i(Tr)$ is a $j$-th subclass of the class $Tr$, which has a cardinality of $i$ and $C_{|Tr|}^i$ is a binomial coefficient, which is equal to a number of all possible unique subsets of the set $M$, which have a cardinality $i$; and where an attribute (property or method) $m$ is defined as $m \in M$.

Now, let us construct the concept lattice for the formal context $W_1$ using the Table 1. Analyzing the results, depicted in Figure 1, we can see, that the constructed concept lattice has a big size and contains all formally possible subclasses of the class $Tr$ and all possible formal concepts of the context $W_1$. Considering the constructed concept lattice, we can ask a question about the semantic consistency of all constructed subclasses of the class $Tr$, as it was done in [28]. To clarify the problem and then answer this question, let us consider in more detail the internal structure of the class $Tr$ and how it is related to the semantic consistency of its subclasses.

**Table 1**

Formal context, which defines all possible subclasses of the class $Tr$.

| | $W_1$ | **Subclasses** | | | | | |
|---|---|---|---|---|---|---|---|
| | | $SC_1^0(Tr)$ | $SC_1^1(Tr)$ | $SC_2^1(Tr)$ | $SC_3^1(Tr)$ | ⋯ | $SC_1^8(Tr)$ |
| **Properties and methods** | $Tr.p_1$ | + | - | - | - | | + |
| | $Tr.p_2$ | - | + | - | - | | + |
| | $Tr.p_3$ | - | - | + | - | | + |
| | $Tr.p_4$ | - | - | - | + | | + |
| | $Tr.f_1$ | - | - | - | - | ⋯ | + |
| | $Tr.f_2$ | - | - | - | - | | + |
| | $Tr.f_3$ | - | - | - | - | | + |
| | $Tr.f_4$ | - | - | - | - | | + |

According to the definition of the homogeneous class of objects, the class $Tr$ consists of a collection of properties $P(Tr) = \{Tr.p_1, Tr.p_2, Tr.p_3, Tr.p_4\}$ called a specification, and a collection of methods $F(Tr) = \{Tr.f_1, Tr.f_2, Tr.f_3, Tr.f_4\}$ called a signature. Analyzing definitions of properties and methods of the class $Tr$, we can find some internal dependencies among them. It is a common practice for many object-oriented programming languages as well as knowledge representation models to define some properties and methods of a class using for this purpose other properties and (or) methods of the class. Such practice allows us to avoid code duplication and provides instead of it code reusability. However, it creates internal dependencies, which help to describe the modeled instance more precisely to the corresponding entity from a particular domain. In addition, such dependencies

are important for the decomposition of classes because they define appropriate constraints for the properties and methods of a class. Furthermore, since not all formally possible subclasses of a class are semantically consistent, i.e., only some of them do not conflict with constraints imposed by the dependencies, the decomposition of a class as the construction of all its subclasses is based on such dependencies.

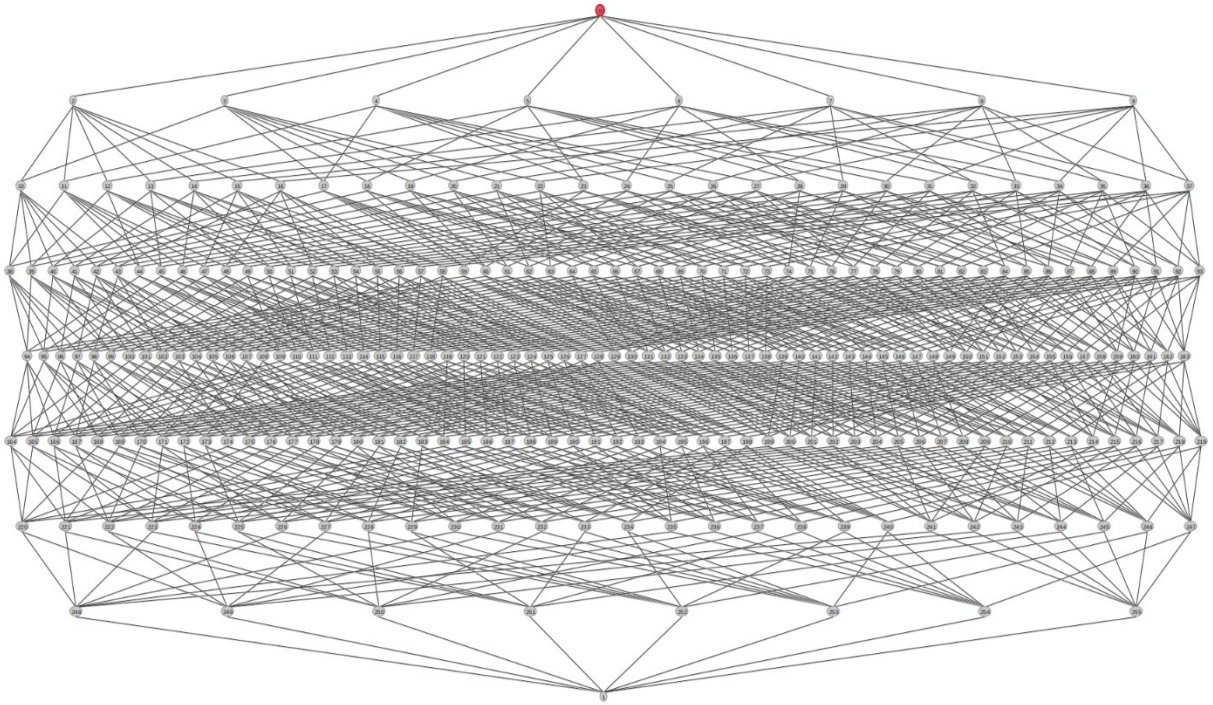

**Figure 1**: Concept lattice of the formal context $W_1$.

Let us consider examples of semantically consistent and inconsistent subclasses of the class $Tr$, to understand the problem more specifically. One of the semantically consistent subclasses of the class $Tr$ is the subclass $SC_{16}^{3}(Tr) = \{Tr.p_1, Tr.f_1, Tr.f_2\}$ because the property $Tr.p_1$ is defined independently from other properties and methods, and it is required for the execution of methods $Tr.f_1$ and $Tr.f_2$, which are determined based on this property. In other words, subclass $SC_{16}^{3}(Tr)$ defines a point on a plane with the ability to get and set its coordinates. One of the semantically inconsistent subclass of the class $Tr$ is a subclass $SC_{17}^{3}(Tr) = \{Tr.p_1, Tr.f_1, Tr.f_3\}$ because as in the previous case, the property $Tr.p_1$ is defined independently from other properties and methods, and it is required for the invocation of the method $Tr.f_1$, however, the correct invocation of the method $Tr.f_3$ demands one more property similar to $Tr.p_1$. In other words, subclass $SC_{17}^{3}(Tr)$ defines a point on a plane with the ability to get its coordinates, but the invocation of the method $Tr.f_3$, which computes the distance between two points on a plane, will cause an error because the subclass $SC_{17}^{3}(Tr)$ determines only one point on a plane. Therefore, it is inconsistent one. Using this fact, we can conclude that the constructed concept lattice of the formal context $W_1$ contains semantically consistent concepts, as well as inconsistent ones. This fact is important for knowledge retrieval since it is avoiding the consideration of semantically inconsistent subclasses and reduce the search space.

To formalize the internal dependencies of a class, concepts of *structural* and *functional atoms*, as well as *structural* and *functional molecules* of the homogeneous class of objects, were introduced in [28]. Since properties and methods of a class can be defined independently of other properties and (or) methods of the class, as well as using them, they are similar to chemical atoms and molecules. Indeed, independent properties and methods are similar to atoms, which are the smallest indivisible particles,

while dependent ones are similar to molecules, which are groups of atoms or smaller molecules somehow connected with each other. As the properties of a class define its structure, while the methods define its behavior, the corresponding atoms and molecules of the class can be classified as structural and functional ones. Let us consider the definitions of both kinds of atoms and molecules of a class, as well as some of their examples, using the class $Tr$.

**Definition 8.** *Structural atom of a homogeneous class of objects* $T$ *is a singleton collection* $SA_i(T) = \{T.p_i\}$, *where* $T.p_i \in P(T)$ *is a property defined without using any other properties and (or) methods of the class* $T$, *where* $P(T)$ *is its specification.*

To analyze the specification of the class $Tr$, we can observe, that quantitative properties $Tr.p_1$, $Tr.p_2$, and $Tr.p_3$, which mean the vertices of a triangle, are defined without usage of any other property or method of the class $Tr$. Therefore, these three properties define structural atoms $SA_1(Tr)$, $SA_2(Tr)$, and $SA_3(Tr)$, respectively, i.e.

$$SA_1(Tr) = \{Tr.p_1\},\ SA_2(Tr) = \{Tr.p_2\},\ SA_3(Tr) = \{Tr.p_3\}.$$

**Definition 9.** *Functional atom of a homogeneous class of objects* $T$ *is singleton collection* $FA_i(T) = \{T.f_i\}$, *where* $T.f_i \in F(T)$ *is a method defined without using any other properties and (or) methods of the class* $T$, *where* $F(T)$ *is its signature.*

The signature of class $Tr$ does not contain any methods defined independently from other properties and methods.

**Definition 10.** *A functional molecule of a homogeneous class of objects* $T$ *is a collection* $FM_i(T) = (T.f_i, \{T.x_{j_1}, ..., T.x_{j_n}\})$ *where* $T.f_i \in F(T)$, $1 \le i \le |F(T)|$ *is a method defined based on the other methods and (or) properties* $\{T.x_{j_1}, ..., T.x_{j_n}\} \subseteq P(T) \cup F(T)$ *which form structural and (or) functional atoms, and are parts of smaller molecules of the class* $T$, *where* $1 \le j_1 \le ... \le j_n \le |P(T) \cup F(T)|$, *and* $P(T)$ *is a specification of the class of objects* $T$, *while* $F(T)$ *is its signature.*

To analyze the structure and behavior of the class $Tr$, we can observe that methods $Tr.f_1$ and $Tr.f_2$, which get and set the coordinates of vertices of a triangle, operate by a particular vertex of the figure. Thus, they define functional molecules $FM_1(Tr)$, and $FM_2(Tr)$, i.e.

$$FM_1(Tr) = (Tr.f_1, \{Tr.p_1\}, \{Tr.p_2\}, \{Tr.p_3\}),\ FM_2(Tr) = (Tr.f_2, \{Tr.p_1\}, \{Tr.p_2\}, \{Tr.p_3\}).$$

In addition, method $Tr.f_3$, which computes the length of a particular side of a triangle, uses a corresponding pair of its vertices. Therefore, it determines a functional molecule $FM_3(Tr)$, i.e.

$$FM_3(Tr) = (Tr.f_3, \{Tr.p_1, Tr.p_2\}, \{Tr.p_1, Tr.p_3\}, \{Tr.p_2, Tr.p_3\}).$$

And finally, method $Tr.f_4$, which calculates the perimeter of a triangle, uses methods $Tr.f_3$ and $Tr.f_1$ to compute the length of each figure's side. As the result, it defines a complex structural molecule $FM_4(Tr)$, which includes the elements of smaller molecules $FM_3(Tr)$ and $FM_1(Tr)$, i.e.

$$FM_4(Tr) = (Tr.f_4, \{Tr.f_3, Tr.f_1, Tr.p_1, Tr.p_2, Tr.p_3\}).$$

**Definition 11.** *A structural molecule of a homogeneous class of objects* $T$ *is a collection* $SM_i(T) = (T.p_i, \{T.x_{j_1}, ..., T.x_{j_n}\})$ *where* $T.p_i \in P(T)$, $1 \le i \le |F(T)|$ *is a property defined based on the other properties and (or) methods* $\{T.x_{j_1}, ..., T.x_{j_n}\} \subseteq P(T) \cup F(T)$, *which form structural*

*and (or) functional atoms, and are parts of smaller molecules of the class* $T$*, where* $1 \le j_1 \le ... \le j_n \le |P(T) \cup F(T)|$*, and* $P(T)$ *is a specification of the class of objects* $T$*, while* $F(T)$ *is its signature.*

The class $T$ has qualitative property $Tr.p_4$, which means the satisfiability of the triangle inequality, and uses methods $Tr.f_3$ and $Tr.f_1$ to compute the length of each figure's side. Hence, it determines a complex structural molecule $SM_4(Tr)$, which includes the elements of smaller molecules $FM_3(Tr)$ and $FM_1(Tr)$, i.e.

$$SM_4(Tr) = (Tr.p_4, \{Tr.f_3, Tr.f_1, Tr.p_1, Tr.p_2, Tr.p_3\}).$$

Since all molecules contain a property or method which is dependent on all other properties and (or) methods of the molecule, we can define a concept of dependency root, which describes such elements.

**Definition 12.** *The dependency root of the molecule* $M_i(T) = (T.a_i, \{T.x_{j_1}, ..., T.x_{j_n}\})$ *of a homogeneous class of objects* $T$ *is a property or method* $T.a_i$ *which is defined based on other properties and (or) methods of the class* $T$*, which are atoms or parts of smaller molecules.*

All detected internal dependencies within the class $Tr$ describe some semantic connections among the different properties and methods of the class, which express the internal nature of the modeled entity from a particular domain if such a model is correct.

**Definition 13.** *Internal semantic dependencies of a homogeneous class of objects* $T$*, which defines type of objects* $t$*, is a set of structural and functional atoms and molecules of the class* $T$*, i.e.*

$$ISD(T) = \{SA_1(T), ..., SA_n(T), FA_1(T), ..., FA_m(T), SM_1(T), ..., SM_w(T), FM_1(T), ..., FM_q(T)\},$$

*where* $SA_{i_1}(T)$*,* $i_1 = \overline{1,n}$ *and* $FA_{j_1}(T)$*,* $j_1 = \overline{1,m}$ *are structural and functional atoms of the class* $T$*, while* $SM_{i_2}(T)$*,* $i_2 = \overline{1,w}$ *and* $FM_{j_2}(T)$*,* $j_2 = \overline{1,q}$ *are its structural and functional molecules respectively.*

All considered atoms and molecules of the class $Tr$ define its internal semantic dependencies, which can be represented in the following way:

$$ISD(Tr) = \{SA_1(Tr), SA_2(Tr), SA_3(Tr), SM_4(Tr), FM_1(Tr), FM_2(Tr), FM_3(Tr), FM_4(Tr)\}.$$

Let us construct the concept lattice of all internal semantic dependencies of the class $Tr$. For this purpose, let us define the formal context $W_2 = (G = ISD(Tr), M = P(Tr) \cup F(Tr), I : G \times M)$ using the corresponding cross table. Since, each of the functional molecules $FM_1(Tr)$, $FM_2(Tr)$, and $FM_3(Tr)$ has three different contexts within the class $Tr$, we let us split them onto separate conditions. We colored cells of Table 2, which means dependency roots of molecules of the class $Tr$, using the gray color. Now let us construct the concept lattice for the formal context $W_2$, using Table 2. Analyzing the results, depicted in Figure 2, we can see that constructed concept lattice contains $22$ nodes, which means formal concepts, however not all of them are semantically consistent ones. We used the green border to highlight the consistent concepts, which do not contradict any internal semantic dependency of the class $Tr$. Indeed, if we consider, for example, the concept $15$, which defined as follows

$$(I = \{Tr.f_3, Tr.p_1\},\ E = \{FM_{31}(Tr), FM_{32}(Tr), FM_4(Tr), SM_4(Tr)\}),$$

we can see that the intent $I = \{Tr.f_3, Tr.p_1\}$ of the concept is semantically inconsistent because it contradicts the internal semantic dependencies defined by functional molecules $FM_{31}(Tr)$,

$FM_{32}(Tr)$, and $FM_{33}(Tr)$. In other words, the concept 14 defines a point on a plane, however, it has a method $Tr.f_3$, which computes the distance between two points on a plane, but the concept defines only one point, that means if we invoke this method, it will raise an error.

**Table 2**
Formal context, which defines internal semantic dependencies of the class $Tr$.

| | $W_2$ | Properties and methods | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | $Tr.p_1$ | $Tr.p_2$ | $Tr.p_3$ | $Tr.p_4$ | $Tr.f_1$ | $Tr.f_2$ | $Tr.f_3$ | $Tr.f_4$ |
| Atoms and molecules | $SA_1(Tr)$ | + | - | - | - | - | - | - | - |
| | $SA_2(Tr)$ | - | + | - | - | - | - | - | - |
| | $SA_3(Tr)$ | - | - | + | - | - | - | - | - |
| | $SM_4(Tr)$ | + | + | + | + | + | - | + | + |
| | $FM_{11}(Tr)$ | + | - | - | - | + | - | - | - |
| | $FM_{12}(Tr)$ | - | + | - | - | + | - | - | - |
| | $FM_{13}(Tr)$ | - | - | + | - | + | - | - | - |
| | $FM_{21}(Tr)$ | + | - | - | - | - | + | - | - |
| | $FM_{22}(Tr)$ | - | + | - | - | - | + | - | - |
| | $FM_{23}(Tr)$ | - | - | + | - | - | + | - | - |
| | $FM_{31}(Tr)$ | + | + | - | - | - | - | + | - |
| | $FM_{32}(Tr)$ | + | - | + | - | - | - | + | - |
| | $FM_{33}(Tr)$ | - | + | + | - | - | - | + | - |
| | $FM_4(Tr)$ | + | + | + | - | + | - | + | + |

Summarizing all noted above, we can conclude that classical methods of formal concept analysis have some gaps related to the semantic consistency of formal concepts, which are constructed within a formal context of decomposition of homogeneous classes of objects. Since, the definition of a formal context does not consider the internal semantic dependencies of a class, defined by its atoms and molecules, the knowledge retrieval or reasoning within a corresponding concept lattice, constructed using such formal context, becomes inconsistent too because its result can contain inconsistent concepts.

## 4. Decomposition of Classes and Objects

Since a homogeneous class of objects consist of structural and functional molecules, which define the restrictions over the class specification and signature, the decomposition of the class can be considered as the constraint satisfaction problem (CSP) [28]. According to [3, 12, 19], the CSP can be defined as a tuple $(X, D, C)$, where $X = \{x_1, ..., x_n\}$, $n > 0$ is a finite sequence of variables, $D = \{d_1, ..., d_n\}$ is a set of domains, and $C = \{c_1, ..., c_m\}$, $m > 0$ is a finite set of constraints. Variables from the set $X$ are defined on domains from the set $D$, i.e. $x_1 \to d_1, ..., x_n \to d_n$, and each domain defines a range of values for the respective variable, i.e. $d_i = \{v_1, ..., v_k\}$, $i = \overline{1,n}$, $k > 0$. Every constraint from the set $C$ is defined as a pair $c_j = (S_j, R_j)$, $j = \overline{1,m}$, where $S_j = (x_{j_1} \to d_{j_1}, ..., x_{j_w} \to d_{j_w})$ is a scope of the constraint and $R_j \in d_{j_1} \times ... \times d_{j_w}$ is a relation de-

fined over the $S_j$, $0 < j_1 < ... < j_w \le n$, and $0 < w \le n$ is the arity of the constraint $c_j$. The tuple $(y_{i_1} : x_1 \to d_1, ..., y_{i_w} : x_w \to d_w)$ satisfies the constraint $c_j = (S_j, R_j)$ on the variables $x_1, ..., x_w$ if and only if $(y_{i_1}, ..., y_{i_w}) \in S_j$. If the tuple $(y_{i_1} : x_1 \to d_1, ..., y_{i_w} : x_w \to d_w)$ satisfies $\forall c_j \in C$, then it is a solution of the CSP.

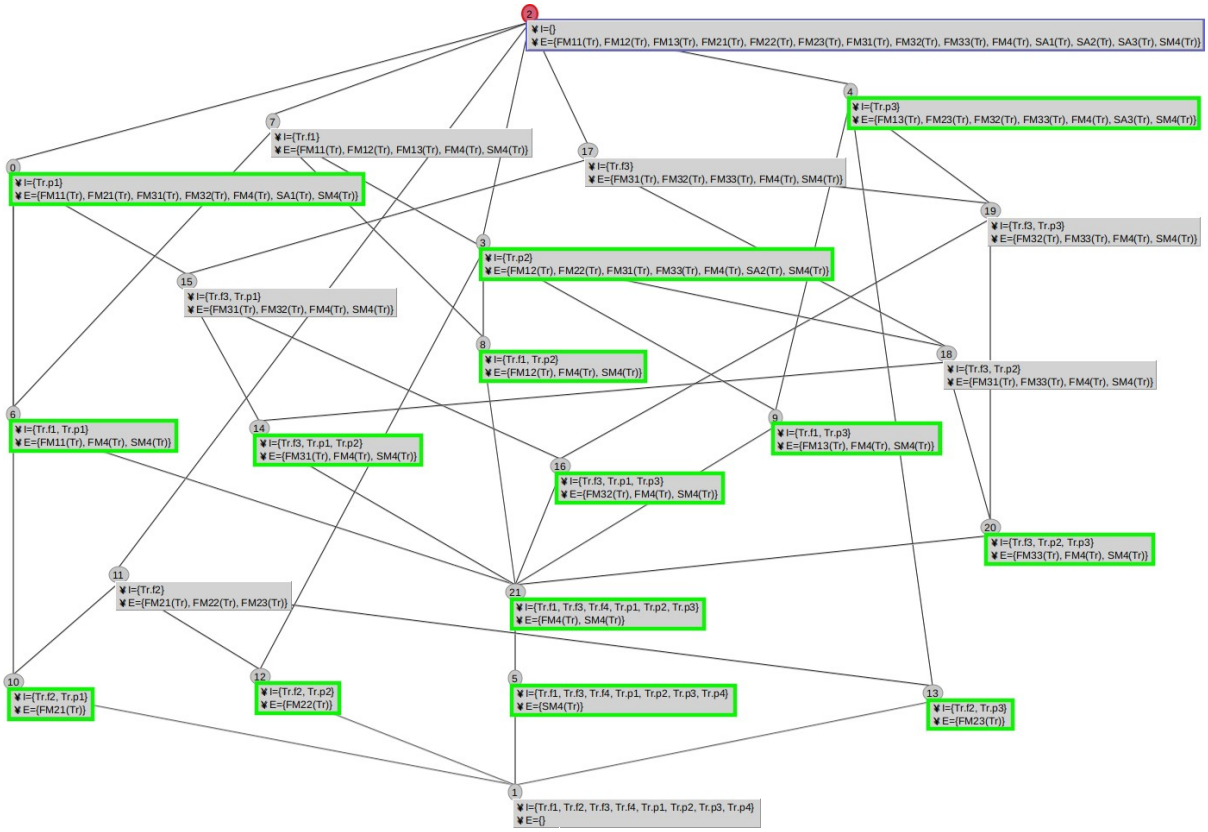


**Figure 2:** Concept lattice of the formal context $W_2$.

However, a particular subclass $T_i \subseteq T$ of a homogeneous class of objects $T$ can satisfy or not satisfy a constraint defined by a molecule $M_j(T)$. Therefore, in contrast to the classical definition of the CSP, the constraint defined by the molecule $M_j(T)$ is applicable only to some subclasses of the class $T$. For example, the constraint defined by functional molecule $FM_4(Tr)$ is not applicable to any subclass of the class $Tr$, which have a cardinality lower than the molecule itself. In such cases, we can conclude that the subclass does not contradict the constraint defined by the molecule, since the constraint is not applicable to it. To summarize these facts, let us introduce the following definition.

**Definition 14.** *A subclass $T_i \subseteq T$ of a homogeneous class of objects $T$ does not contradict molecular internal semantic dependency $M_j(T) = (T.a_j, \{T.x_{k_1}, ..., T.x_{k_m}\})$, if and only if one of the following conditions is true:*

1. *it contains all elements of the molecule, i.e. $\forall x \in M_j(T)$, $x \in P(T_i) \cup F(T_i)$;*
2. *it does not contain any element of the molecule, i.e. $\forall x \in M_j(T)$, $x \notin P(T_i) \cup F(T_i)$;*
3. *it does not contain the dependency root of the molecule, but it contains some of its other elements, i.e. $T.a_j \notin P(T_i) \cup F(T_i)$, and $\exists x \in \{T.x_{k_1}, ..., T.x_{k_m}\}$, $x \in P(T_i) \cup F(T_i)$;*

*where $T.a_j \in P(T) \cup F(T)$, $1 \le j \le |P(T) \cup F(T)|$ is a property or method defined based on the other properties and (or) methods $\{T.x_{k_1}, ..., T.x_{k_m}\} \subseteq P(T) \cup F(T)$ of the class $T$, where*

$1 \le k_1 \le ... \le k_m \le |P(T) \cup F(T)|$, *and* $P(T)$, $P(T_i)$ *and* $F(T)$, $F(T_i)$ *are specifications and signatures of the class* $T$ *and* $T_i$ *respectively.*

Using this notion, we can define the decomposition of the homogeneous classes of objects.

**Definition 15.** *A decomposition of a homogeneous class of objects* $T$*, which defines a type of objects* $t$*, is a set of semantically consistent subclasses* $D(T) = \{T_1 \subseteq T, ..., T_n \subseteq T\}$*, where subclasses* $T_1, ..., T_n$ *do not contradict any molecular internal semantic dependency of the class* $T$*.*

Now, let us compute the full decomposition of the homogeneous class of objects $Tr$, using the corresponding algorithm, which was proposed in [28], and the set of internal semantic dependencies $ISD(Tr)$, which defines a collection of decomposition constraints. As the result of decomposition we obtained the collection of 49 semantically consistent subclasses of the class $Tr$, where three subclasses of the cardinality of 1, i.e.

$$SC_1^1(Tr) = (Tr.p_1),\ SC_2^1(Tr) = (Tr.p_2),\ SC_3^1(Tr) = (Tr.p_3),$$

nine subclasses of the cardinality of 2, i.e.

$$SC_1^2(Tr) = (Tr.p_1, Tr.p_2),\ SC_2^2(Tr) = (Tr.p_1, Tr.p_3),\ SC_3^2(Tr) = (Tr.p_2, Tr.p_3),$$
$$SC_4^2(Tr) = (Tr.p_1, Tr.f_1),\ SC_5^2(Tr) = (Tr.p_2, Tr.f_1),\ SC_6^2(Tr) = (Tr.p_3, Tr.f_1),$$
$$SC_7^2(Tr) = (Tr.p_1, Tr.f_2),\ SC_8^2(Tr) = (Tr.p_2, Tr.f_2),\ SC_9^2(Tr) = (Tr.p_3, Tr.f_3),$$

thirteen subclasses of the cardinality of 3, i.e.

$$SC_1^3(Tr) = (Tr.p_1, Tr.p_2, Tr.p_3),\ SC_2^3(Tr) = (Tr.p_1, Tr.p_2, Tr.f_1),\ SC_3^3(Tr) = (Tr.p_1, Tr.p_3, Tr.f_1),$$
$$SC_4^3(Tr) = (Tr.p_2, Tr.p_3, Tr.f_1),\ SC_5^3(Tr) = (Tr.p_1, Tr.p_2, Tr.f_2),\ SC_6^3(Tr) = (Tr.p_1, Tr.p_3, Tr.f_2),$$
$$SC_7^3(Tr) = (Tr.p_2, Tr.p_3, Tr.f_2),\ SC_8^3(Tr) = (Tr.p_1, Tr.f_1.Tr.f_2),\ SC_9^3(Tr) = (Tr.p_2, Tr.f_1, Tr.f_2),$$
$$SC_{10}^3(Tr) = (Tr.p_3, Tr.f_1, Tr.f_2),\ SC_{11}^3(Tr) = (Tr.p_1, Tr.p_2, Tr.f_3),\ SC_{12}^3(Tr) = (Tr.p_1, Tr.p_3, Tr.f_3),$$
$$SC_{13}^3(Tr) = (Tr.p_2, Tr.p_3, Tr.f_3),$$

twelve subclasses of the cardinality of 4, i.e.

$$SC_1^4(Tr) = (Tr.p_1, Tr.p_2, Tr.p_3, Tr.f_1),\ SC_2^4(Tr) = (Tr.p_1, Tr.p_2, Tr.p_3, Tr.f_2),$$
$$SC_3^4(Tr) = (Tr.p_1, Tr.p_2, Tr.f_1, Tr.f_2),\ SC_4^4(Tr) = (Tr.p_1, Tr.p_3, Tr.f_1, Tr.f_2),$$
$$SC_5^4(Tr) = (Tr.p_2, Tr.p_3, Tr.f_1, Tr.f_2),\ SC_6^4(Tr) = (Tr.p_1, Tr.p_2, Tr.p_3, Tr.f_3),$$
$$SC_7^4(Tr) = (Tr.p_1, Tr.p_2, Tr.f_1, Tr.f_3),\ SC_8^4(Tr) = (Tr.p_1, Tr.p_3, Tr.f_1, Tr.f_3),$$
$$SC_9^4(Tr) = (Tr.p_2, Tr.p_3, Tr.f_1, Tr.f_3),\ SC_{10}^4(Tr) = (Tr.p_1, Tr.p_2, Tr.f_2, Tr.f_3),$$
$$SC_{11}^4(Tr) = (Tr.p_1, Tr.p_3, Tr.f_2, Tr.f_3),\ SC_{12}^4(Tr) = (Tr.p_2, Tr.p_3, Tr.f_2, Tr.f_3),$$

six subclasses of the cardinality of 5, i.e.

$$SC_1^5(Tr) = (Tr.p_1, Tr.p_2, Tr.p_3, Tr.f_1, Tr.f_2),\ SC_2^5(Tr) = (Tr.p_1, Tr.p_2, Tr.p_3, Tr.f_1, Tr.f_3),$$
$$SC_3^5(Tr) = (Tr.p_1, Tr.p_2, Tr.p_3, Tr.f_2, Tr.f_3),\ SC_4^5(Tr) = (Tr.p_1, Tr.p_2, Tr.f_1, Tr.f_2, Tr.f_3),$$
$$SC_5^5(Tr) = (Tr.p_1, Tr.p_3, Tr.f_1, Tr.f_2, Tr.f_3),\ SC_6^5(Tr) = (Tr.p_2, Tr.p_3, Tr.f_1, Tr.f_2, Tr.f_3),$$

three subclasses of the cardinality of 6, i.e.

$$SC_1^6(Tr) = (Tr.p_1, Tr.p_2, Tr.p_3, Tr.p_4, Tr.f_1, Tr.f_3),$$
$$SC_2^6(Tr) = (Tr.p_1, Tr.p_2, Tr.p_3, Tr.f_1, Tr.f_2, Tr.f_3),$$
$$SC_3^6(Tr) = (Tr.p_1, Tr.p_2, Tr.p_3, Tr.f_1, Tr.f_3, Tr.f_4),$$

and three subclasses of the cardinality of 7, i.e.

$$SC_1^7(Tr) = (Tr.p_1, Tr.p_2, Tr.p_3, Tr.p_4, Tr.f_1, Tr.f_2, Tr.f_3),$$
$$SC_2^7(Tr) = (Tr.p_1, Tr.p_2, Tr.p_3, Tr.p_4, Tr.f_1, Tr.f_3, Tr.f_4),$$
$$SC_3^7(Tr) = (Tr.p_1, Tr.p_2, Tr.p_3, Tr.f_1, Tr.f_2, Tr.f_3, Tr.f_4).$$

Now let us construct the concept lattice for all semantically consistent subclasses of the class $Tr$. For this purpose, let us consider the formal context $W_3 = (G = D(Tr), M = P(T) \cup F(T), I : G \times M)$ using the structure of semantically consistent subclasses of the class $Tr$ noted above. As we can see, the concept lattice for the formal context $W_3$, depicted in Figure 3, contains 70 formal concepts, while the amount of all semantically consistent subclasses constructed by the decomposition algorithm is equal to 49. It means that some formal concepts in the lattice are semantically consistent, while other ones are inconsistent. For example, the formal concept 44 is semantically inconsistent because its intent contradicts internal semantic dependencies $FM_{31}(Tr)$, $FM_{32}(Tr)$, and $FM_{33}(Tr)$, similarly to the case of concept 15 in formal context $W_2$. It happens because the algorithms for constructing of concept lattices compute the part of extents as the intersection of those extents which can be extracted from the formal context cross table [21]. They do not consider internal semantic dependencies within the classes and objects, consequently, they escape a question about the existence of such concepts within a modeled domain, rather compute only intersection among objects or classes to obtain sets of common attributes as new concepts.

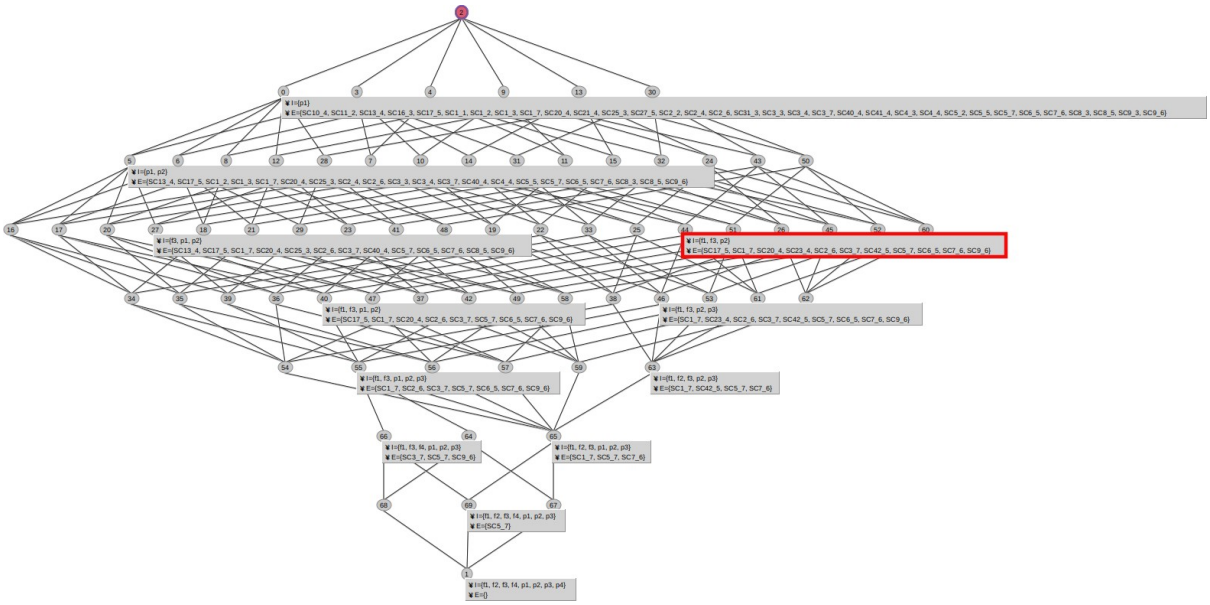

**Figure 3**: Partially annotated concept lattice of the formal context $W_3$.

We think that it is an important restriction for the usage of formal concept analysis, in particular, for knowledge retrieval and reasoning, since there is an ability to retrieve or infer concepts, which are inconsistent, and therefore unreal within a modeled domain.

Let us compare the amount of semantically consistent subclasses of the class $Tr$ with the amount of all possible subclasses, splitting them according to antichains of join-semilattice created by all subclasses.

**Table 3**
Quantitative analysis of subclasses of the class $Tr$.

| Cardinality | 1 | 2 | 3 | 4 | 5 | 6 | 7 | Total |
|---|---|---|---|---|---|---|---|---|
| **Possible subclasses** | 8 | 28 | 56 | 70 | 56 | 28 | 8 | **254** |
| **Consistent subclasses** | 3 | 9 | 13 | 12 | 6 | 3 | 3 | **49** |
| **Decomposition consistency** | 38% | 32% | 23% | 17% | 11% | 11% | 38% | **19%** |

Analyzing Table 1, we can see that among all 254 formally possible nonempty proper subclasses of the homogeneous class of objects $Tr$, only 49, i.e. 19%, are semantically consistent ones, i.e. they do not contradict any of the internal semantic dependencies of the class. This coefficient allows us to estimate how the search space can be reduced by avoiding the consideration of all semantically inconsistent subclasses of the class $Tr$. Therefore, let us introduce the corresponding definition for it.

**Definition 16.** *Decomposition consistency of a homogeneous class of objects* $T$ *is a coefficient* $DC(T)$ *computed in the following way*

$$DC(T) = \frac{|D(T)|}{|PS(T)|-2} \cdot 100\%,$$

*where* $D(T)$ *is a set of all semantically consistent subclasses of the class* $T$*, while* $PS(T)$ *is a power set of its all possible subclasses.*

Since $DC(Tr) \approx 19\%$, it means that we can reduce the knowledge search space for the class $Tr$ approximately by $5.3$ times, i.e. $100\% : 19\% \approx 5.3$.

All data given in Table 1 can be represented graphically, that provides an opportunity to estimate the search space for the knowledge extraction from another perspective. Figure 4 illustrates elements of the power set lattice, where each element is a subclass of the homogeneous class of objects $Tr$.

**Figure 4**: Tower of the subclass lattice of the class $Tr$.

Circles depicted by lime color mean semantically consistent subclasses, while yellow circles with numbers mean a particular antichain of the lattice, or in other words, a set of subclasses of the corresponding cardinality. We also can see, that each element of this lattice, which has a cardinality bigger than 2, and lower than the class itself, can be also decomposed into subclasses, where some of them are semantically consistent, while others are not so. We also depicted in Figure 4 towers of subclass lattice for particular semantically consistent subclasses of cardinality from 2 to 7, which illustrates that the subclass lattice tower of the class $Tr$ contains towers of subclass lattices. The graphical representation of the complete lattice, illustrated in Figure 4, is not a typical or common way to the depiction of lattices, such as the Hasse diagram, for example. However, as you can see the power set lattice of the class $Tr$ contains 256 elements and, as was noted in [32], $n2^{n-1} = 8 \cdot 2^7 = 1024$ connec-

tions, which makes the corresponding Hasse diagram complicated. Instead of this, for the quantitative analysis of semantically consistent subclasses of a particular homogeneous class of objects, we can depict only elements of the lattice's antichains. Since the geometrical form of such representation reminds a tower, we called it a tower of the power set lattice or a tower of subclass lattice.

However, not only classes of objects can be decomposed to extract new knowledge. Since the main purpose of the definition of classes is creating objects, which also can be decomposed based on semantically consistent subclasses constructed by Algorithm 1. Therefore, let us define the notion of object decomposition.

**Definition 17.** *A decomposition of an object* $k$ *of a homogeneous class of objects* $T$*, which defines a type of objects* $t$*, is a set of objects* $D(k) = \{k_1, ..., k_n\}$*, where the object* $k_i$*,* $i = \overline{1,n}$ *belongs to the semantically consistent subclass* $T_i \in D(T)$ *and* $\forall k_i.p, \exists! k.p \mid k_i.p = k.p$*, where* $k_i.p$ *and* $k.p$ *are properties of objects.*

Considering the fact, that objects can exist during the program run-time and can be stored in the database or knowledge base, their structure in each of these states can be different. Based on this idea and using the notion of object decomposition, we developed Algorithm 1 and Algorithm 2 for the decomposition of objects oriented on their run-time representation, and their storage in a database, respectively.

**Algorithm 1.** Run-time decomposition of objects.

**Require:** $O_T$, $SC(T)$

**Ensure:** $O_{SC(T)}$

**1:** $O_{SC(T)} := []$;
**2:** **for** $i = 1, ..., |O_T|$ **do**
**3:** $object := SC(T)()$;
**4:** **for all** $SC(T).p \in P(SC(T))$ **do**
**5:** $object[SC(T).p] := O_T[i].get(SC(T).p)$;
**6:** $O_{SC(T)}$.append($object$);
**7:** **return** $O_{SC(T)}$.

Analyzing Algorithm 1, we can see that it generates the set of objects $O_{SC(T)}$ for the subclass $SC(T)$ constructed during the decomposition of the homogeneous class of objects $T$ based on the set of its objects $O_T$. The algorithm creates a particular object for the subclass $SC(T)$ initializes each its property by corresponding value, which is got from the corresponding object of the class $T$.

**Algorithm 2.** Database-oriented decomposition of objects.

**Require:** $O_T$, $SC(T)$

**Ensure:** $O_{SC(T)}$

**1:** $O_{SC(T)} := []$;
**2:** **for** $i = 1, ..., |O_T|$ **do**
**3:** $object := \{\}$;
**4:** **for all** $SC(T).a \in P(SC(T)) \cup F(SC(T))$ **do**
**5:** **if** is_property($SC(T).a$) **then**
**6:** $object[SC(T).a] := O_T[i].get(SC(T).a)$;
**7:** **if** is_method($SC(T).a$) **and** $SC(T).a()$ **then**

**8:** $object[SC(T).a] := object.[SC(T).a]();$

**9:** $O_{SC(T)}.\text{append}(object);$

**10:** **return** $O_{SC(T)}$.

Algorithm 2 also generates the set of objects $O_{SC(T)}$ for the subclass $SC(T)$ constructed during the decomposition of the homogeneous class of objects $T$ based on the set of its objects $O_T$. However, in contrast to Algorithm 1, it generates an extended representation of objects by storing the results of the invocation of methods, which are available for them in the class $SC(T)$. Since methods of the class can return different values as the result, their correct storage in the database can require additional processing, depending on the type of returned result.

Now let us consider some examples of objects' decomposition, using Algorithm 2. Let us define and decompose a number of objects of the class $Tr$, which represent particular triangles on a plane. For this purpose, let us define the collection of points on a plane via creating objects of the class $Pt$ (see Table 4).

**Table 4**
Points on a plane represented by objects of the class $Pt$.

| | $x$ | $y$ |
|---|---|---|
| $point_1$ | 0 | 0 |
| $point_2$ | 0 | 5 |
| $point_3$ | 2 | 10 |
| $point_4$ | 5 | 9 |
| $point_5$ | 6 | 3 |
| $point_6$ | 8 | 0 |
| $point_7$ | 10 | 2 |

Now let us determine triangles on a plane, creating objects of the class $Tr$ and using the above-defined points as vertices of figures (see Table 5). Since objects are containers, which encapsulate property values of particular instances of a class, all objects of the class $Tr$ are defined by the following four properties: $triangle_i.p_1$, $triangle_i.p_2$, $triangle_i.p_3$, and $triangle_i.p_4$, where $i = \overline{1,8}$. Such structure for objects of the class is commonly-used for their representation in the program run-time. For the storage in a database, we can use results computed by methods $Tr.f_3$ and $Tr.f_4$, where the first one computes the length of the triangle's sides while the second one computes the perimeter of the figure. Consequently, we can consider at least two scenarios of objects' lifetime and their usage, therefore Table 5 contains both representations.

**Table 5**
Triangles on a plane represented by objects of the class $Tr$.

| | $vertex_1$ | $vertex_2$ | $vertex_3$ | $is\_triangle$ | $side_1$ | $side_2$ | $side_3$ | $perimeter$ |
|---|---|---|---|---|---|---|---|---|
| $triangle_1$ | $point_1$ | $point_2$ | $point_6$ | 1 | 5.0 | 6.32 | 6.71 | 18.03 |
| $triangle_2$ | $point_1$ | $point_6$ | $point_7$ | 1 | 6.71 | 3.61 | 8.0 | 18.32 |
| $triangle_3$ | $point_1$ | $point_5$ | $point_6$ | 1 | 10.3 | 6.08 | 6.71 | 23.09 |
| $triangle_4$ | $point_2$ | $point_3$ | $point_4$ | 1 | 5.39 | 2.83 | 5.0 | 13.22 |
| $triangle_5$ | $point_2$ | $point_4$ | $point_5$ | 1 | 5.0 | 1.41 | 6.4 | 12.81 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $triangle_6$ | $point_1$ | $point_3$ | $point_7$ | 1 | 10.2 | 11.66 | 8.0 | 29.86 |
| $triangle_7$ | $point_5$ | $point_6$ | $point_7$ | 1 | 6.08 | 3.61 | 9.49 | 19.18 |
| $triangle_8$ | $point_4$ | $point_5$ | $point_7$ | 1 | 1.41 | 9.49 | 8.94 | 19.84 |

As the result of the full decomposition of the class $Tr$, Algorithm 1 constructed 49 semantically consistent subclasses. It means that if we need to perform a decomposition of all objects of the class $Tr$, we should do it for each constructed subclass. Due to the big number of subclasses, let us consider the decomposition of all objects of the class $Tr$, but only for some of its subclasses, e.g. for $SC_4^2(Tr)$, $SC_1^3(Tr)$, $SC_7^4(Tr)$, and $SC_2^5(Tr)$. According to the definition of the class $Tr$, its subclass $SC_4^2(Tr)$ has the following representation

$$SC_4^2(Tr) = (p_1 = (vertex_1, (v, Pt)),$$
$$f_1 = get\_vertex(tr, (i, \mathbb{Z}^+), Pt)).$$

To decompose objects of the class $Tr$, represented in the Table 5, for its subclass $SC_4^2(Tr)$ we need to set for the Algorithm 2 the following configuration $D(SC_4^2(Tr)) = (O_{Tr}, SC_4^2(Tr))$, where $O_{Tr}$ is a set of all objects of the class $Tr$. As the result, the algorithm constructed objects of the subclass $SC_4^2(Tr)$ (see Table 6). Each of them defines a point on a plane with the ability to get its coordinates as objects of the class $Pt$.

**Table 6**

Objects of the class $SC_4^2(Tr)$.

| | $vertex_1$ |
|---|---|
| $triangle_1$ | $point_1$ |
| $triangle_2$ | $point_1$ |
| $triangle_3$ | $point_1$ |
| $triangle_4$ | $point_2$ |
| $triangle_5$ | $point_2$ |
| $triangle_6$ | $point_1$ |
| $triangle_7$ | $point_5$ |
| $triangle_8$ | $point_4$ |

Subclass $SC_1^3(Tr)$ has the following representation

$$SC_1^3(Tr) = (p_1 = (vertex_1, (v, Pt)),$$
$$p_2 = (vertex_2, (v, Pt)),$$
$$p_3 = (vertex_3, (v, Pt))).$$

To decompose objects of the class $Tr$, represented in the Table 5, for its subclass $SC_1^3(Tr)$ we need to set for the Algorithm 2 the following configuration $D(SC_1^3(Tr)) = (O_{Tr}, SC_1^3(Tr))$. As the result, the algorithm constructed objects of the subclass $SC_1^3(Tr)$ (see Table 7). Each of them defines three points on a plane, where each point is an object of the class $Pt$.

**Table 7**

Objects of the class $SC_1^3(Tr)$.

| | $vertex_1$ | $vertex_2$ | $vertex_3$ |
|---|---|---|---|
| $triangle_1$ | $point_1$ | $point_2$ | $point_6$ |
| $triangle_2$ | $point_1$ | $point_6$ | $point_7$ |
| $triangle_3$ | $point_1$ | $point_5$ | $point_6$ |
| $triangle_4$ | $point_2$ | $point_3$ | $point_4$ |
| $triangle_5$ | $point_2$ | $point_4$ | $point_5$ |
| $triangle_6$ | $point_1$ | $point_3$ | $point_7$ |
| $triangle_7$ | $point_5$ | $point_6$ | $point_7$ |
| $triangle_8$ | $point_4$ | $point_5$ | $point_7$ |

Subclass $SC_7^4(Tr)$ has the following representation

$$\begin{aligned} SC_7^4(Tr) = (&p_1 = (vertex_1, (v, Pt)),\\ &p_2 = (vertex_2, (v, Pt)),\\ &f_1 = get\_vertex(tr, (i, \mathbb{Z}^+), Pt),\\ &f_3 = compute\_side\_length(tr, (vertex_a, Pt), (vertex_b, Pt), \mathbb{R}^+)). \end{aligned}$$

To decompose objects of the class $Tr$, represented in the Table 5, for its subclass $SC_7^4(Tr)$ we need to set for the Algorithm 2 the following configuration $D(SC_7^4(Tr)) = (O_{Tr}, SC_7^4(Tr))$. As the result, the algorithm constructed objects of the subclass $SC_7^4(Tr)$ (see Table 8). Each of them defines two points on a plane with the ability to get their coordinates as objects of the class $Pt$, as well as to compute the distance between them.

**Table 8**

Objects of the class $SC_7^4(Tr)$.

| | $vertex_1$ | $vertex_2$ | $side_1$ |
|---|---|---|---|
| $triangle_1$ | $point_1$ | $point_2$ | 5.0 |
| $triangle_2$ | $point_1$ | $point_6$ | 6.71 |
| $triangle_3$ | $point_1$ | $point_5$ | 10.3 |
| $triangle_4$ | $point_2$ | $point_3$ | 5.39 |
| $triangle_5$ | $point_2$ | $point_4$ | 5.0 |
| $triangle_6$ | $point_1$ | $point_3$ | 10.2 |
| $triangle_7$ | $point_5$ | $point_6$ | 6.08 |
| $triangle_8$ | $point_4$ | $point_5$ | 1.41 |

Subclass $SC_2^5(Tr)$ has the following representation

$$SC_2^5(Tr) = (p_1 = (vertex_1, (v, Pt)),$$
$$p_2 = (vertex_2, (v, Pt)),$$
$$p_3 = (vertex_3, (v, Pt)),$$
$$f_1 = get\_vertex(tr, (i, \mathbb{Z}^+), Pt),$$
$$f_3 = compute\_side\_length(tr, (vertex_a, Pt), (vertex_b, Pt), \mathbb{R}^+)).$$

To decompose objects of the class $Tr$, represented in the Table 5, for its subclass $SC_2^5(Tr)$ we need to set for the Algorithm 2 the following configuration $D(SC_2^5(Tr)) = (O_{Tr}, SC_2^5(Tr))$. As the result, the algorithm constructed objects of the subclass $SC_2^5(Tr)$ (see Table 9). Each of them defines three points on a plane with the ability to get their coordinates as objects of the class $Pt$, as well as to compute the distance between any two of them.

**Table 9**

Objects of the class $SC_2^5(Tr)$.

| | $vertex_1$ | $vertex_2$ | $vertex_3$ | $side_1$ | $side_2$ | $side_3$ |
|---|---|---|---|---|---|---|
| $triangle_1$ | $point_1$ | $point_2$ | $point_6$ | 5.0 | 6.32 | 6.71 |
| $triangle_2$ | $point_1$ | $point_6$ | $point_7$ | 6.71 | 3.61 | 8.0 |
| $triangle_3$ | $point_1$ | $point_5$ | $point_6$ | 10.3 | 6.08 | 6.71 |
| $triangle_4$ | $point_2$ | $point_3$ | $point_4$ | 5.39 | 2.83 | 5.0 |
| $triangle_5$ | $point_2$ | $point_4$ | $point_5$ | 5.0 | 1.41 | 6.4 |
| $triangle_6$ | $point_1$ | $point_3$ | $point_7$ | 10.2 | 11.66 | 8.0 |
| $triangle_7$ | $point_5$ | $point_6$ | $point_7$ | 6.08 | 3.61 | 9.49 |
| $triangle_8$ | $point_4$ | $point_5$ | $point_7$ | 1.41 | 9.49 | 8.94 |

As we can see, the decomposition of objects of the class $Tr$ is based on the decomposition of the class itself. It guarantees that all generated objects of any semantically consistent subclass of the class will be also semantically consistent ones. Object decomposition allows us to generate instances for the subclasses created during the class decomposition, and to perform knowledge extraction on another representation level.

## 5. Knowledge Extraction and Retrieval

One of the approaches to knowledge retrieval was proposed in [24], according to which a formal context can be matched by its formal sub-context constructed using three main kinds of incidence relations $\{gJm_s \mid \exists m \in m_s, gIm\}$, $\{gJm_s \mid \forall m \in m_s, gIm\}$, and $\{gJm_s \mid \mid m \in m_s, gIm \mid / \mid m_s \mid \quad \alpha_{m_s}\}$, that allows the cauterization of the formal context. However, such an approach requires constructing additional concept lattices, to perform their matching with the main concept lattice creating clusters, that can affect the performance of knowledge extraction. Therefore, let us consider another approach.

As we can see, the algorithm for decomposition of homogeneous classes of objects, proposed in [28], can be used for knowledge extraction of semantically consistent subclasses of a class. However, it also can be adapted for knowledge retrieval, by adding additional filtration parameters, which will provide new functional opportunities for conceptual knowledge retrieval and speed up the retrieval process itself. For this purpose, let us add the parameter

$$N = \left[ n_1, ..., n_k \right], \; k = \mid P(T) \cup F(T) \mid -1$$

which means the list of subclass cardinalities and allows the algorithm to construct only those semantically consistent subclasses of a class $T$, whose cardinality is matched with one of the list $N$. It allows us to further reduce the search space for the algorithm if we know the exact cardinality of the semantically consistent subclasses of the class $T$, that we want to retrieve. In addition, we can add parameter

$$Q_a = \left[include = \left[T.a_{i_1},...,T.a_{i_w}\right], exclude = \left[T.a_{j_1},...,T.a_{j_q}\right]\right],$$

which means the attribute query and allows the algorithm to construct only those semantically consistent subclasses of the class $T$, whose contain and do not contain properties and (or) methods from the include and exclude list respectively. It also helps the algorithm to reduce the number of constructed subclasses, if we know useful information about them, i.e. which properties and (or) methods they should and should not contain. Finally, we can add parameter

$$Q_d = \left[include = \left[d_{i_1}(T),...,d_{i_v}(T)\right], exclude = \left[d_{j_1}(T),...,d_{j_m}(T)\right]\right],$$

which means the dependency query and allows the algorithm to construct only those semantically consistent subclasses of the class $T$, whose contain and do not contain properties and (or) methods that are parts of internal semantic dependencies, from include and exclude lists respectively. Similar to the attributes, it also helps the algorithm to reduce the number of constructed subclasses, if we know other useful information about them, i.e. elements of which structural and (or) functional molecules they should and should not contain. Parameters $Q_a$ and $Q_d$ are filters, which allow us to retrieve semantically consistent subclasses of the class $T$ according to particular structural and behavior features. Using all these filtration parameters, we can improve the algorithm for the decomposition of homogeneous classes of objects in the following way.

**Algorithm 3.** Decomposition of homogeneous classes of objects.

**Require:** $T$, $C$, $N$, $Q_a$, $Q_d$
**Ensure:** $D(T)$

```
1:  D(T) := {};
2:  for n ∈ N do
3:      t := {};
4:      for i = 1,...,2^n − 1 do
5:          if binary(i).count(1) = i then
6:              for a_j ∈ T, j = 1,...,|T| do
7:                  if (i & (1 << j)) > 0 then
8:                      t.add(a_j);
9:              satisfy := true;
10:             for all c ∈ C do
11:                 if not satisfy_constraint(t, c) then
12:                     satisfy := false;
13:                     break;
14:             if satisfy then
15:                 if satisfy_query(t, Q_a) and satisfy_query(t, Q_d) then
16:                     D(T).add(t);
17:             t := {};
18: return D(T).
```

As we can see, Algorithm 3 performs the decomposition of the homogeneous class of objects $T$, resolving the corresponding constraint satisfaction problem (CSP), using the set of its internal semantic dependencies $C = ISD(T)$, as well as the list of subclass cardinalities $N$, attribute query $Q_a$, and dependency query $Q_d$. Using the list of subclass cardinalities $N$, the algorithm resolves the CSP only for those subclasses, whose cardinality is matched with one of the list $N$. The set of constraints $C$ is used by the procedure $\text{satisfy\_constraint}(t, c)$ to verify the satisfiability of the constraint $c \in C$ for the subclass $t \subseteq T$, if the constraint is applicable to the subclass. In other words, the procedure $\text{satisfy\_constraint}(t, c)$ resolves the CSP for particular subclass of the class $T$ and if the CSP is satisfiable, then the subclass is semantically consistent. It allows the algorithm constructs only semantically consistent subclasses of the class $T$. For each such subclass of the class $T$, the algorithm performs the additional filtration according to attribute query $Q_a$ and dependency query $Q_d$, using for this purpose the procedure $\text{satisfy\_query}(t, Q)$. As the result, the algorithm constructs all semantically consistent subclasses of the homogeneous class of objects $T$, which have a certain cardinality and satisfy the attribute and dependency restrictions, if such subclasses exist. In general, Algorithm 3 performs two tasks, firstly, it extracts the conceptual knowledge via decomposition of homogeneous classes of objects onto the set of semantically consistent subclasses, and secondly, it retrieves the particular subclasses, which satisfy the corresponding restrictions.

**Procedure 1.** $\text{satisfy\_constraint}(t, c)$

**Input:** $t$, $c$
**Output:** satisfy $\in$ {**true**, **false**, **none**}

```
1:  satisfy := none;
2:  if c[0]∈t then
3:     for i = 1,...,|c| do
4:        for j = 1,...,|c[i]| do
5:           if c[i][j]∈t then
6:              satisfy := true;
7:           else
8:              satisfy := false;
9:              break;
10:       if satisfy then
11:          return satisfy
12: return satisfy.
```

**Procedure 2.** $\text{satisfy\_query}(t, Q)$

**Input:** $t$, $Q$
**Output:** satisfy $\in$ {**true**, **false**}

```
1:  for q∈Q[include] do
2:     if q∉t then
3:        return false;
4:  for q∈Q[exclude] do
5:     if q∈t then
6:        return false;
7:  return true.
```

Consequently, there are two different scenarios for the organization of conceptual knowledge retrieval. In the first case, we can construct all possible semantically consistent subclasses of the class $T$ and then store them in a database, using for this object-relational mapping. Indeed, according to [1-2], each class will be mapped into the database as a corresponding table, where a particular subclass property will be mapped in the corresponding column of the table. Following that, we can use SQL to perform the information retrieval. However, such mapping is applicable only to properties of the class that restricts the usability of the approach because methods can be parts of structural and functional molecules of the class. In the second case, we can perform the information retrieval on the fly, via dynamic filtering of constructed semantically consistent subclasses. To perform the filtering, we can use any query language, which is applicable for the querying over homogeneous classes of objects. However, in this case, we need either develop our own processor or adapt one of the appropriate ones to convert the selected query language to object-oriented structures.

To filter semantically consistent subclasses during the retrieval stage, we propose to use attribute query $Q_a$ and dependency query $Q_d$, which describe the inclusion of desired attributes and dependencies, as well as the exclusion of undesired ones. Let us consider a few examples of dynamic knowledge retrieval using the homogeneous class of objects $Tr$ defined above. Suppose we want to

retrieve all semantically consistent subclasses of the class $Tr$, which have a cardinality from 4 to 6, and contain attributes $Tr.p_1$ and $Tr.f_1$, for this purpose we need to set for Algorithm 3 the following configuration:

$$D_1 = \left( Tr,\ ISD(Tr),\ N = [4,5,6],\ Q_a = \left[ include = \left[ Tr.p_1, Tr.f_1 \right],\ exclude = \left[\ \right] \right], \right.$$
$$\left. Q_d = \left[ include = \left[\ \right],\ exclude = \left[\ \right] \right] \right).$$

As the result, we received the following subclasses: $SC_1^4(Tr)$, $SC_3^4(Tr)$, $SC_4^4(Tr)$, $SC_7^4(Tr)$, $SC_8^4(Tr)$, $SC_1^5(Tr)$, $SC_2^5(Tr)$, $SC_4^5(Tr)$, $SC_5^5(Tr)$, $SC_1^6(Tr)$, $SC_2^6(Tr)$, and $SC_3^6(Tr)$. Suppose we want to retrieve all semantically consistent subclasses of the class $Tr$, which have the same cardinality as previously, and do not contain attributes $Tr.f_2$ and $Tr.f_4$, for this purpose we need to set for Algorithm 3 the following configuration:

$$D_2 = \left( Tr,\ ISD(Tr),\ N = [4,5,6],\ Q_a = \left[ include = \left[\ \right],\ exclude = \left[ Tr.f_2, Tr.f_4 \right] \right], \right.$$
$$\left. Q_d = \left[ include = \left[\ \right],\ exclude = \left[\ \right] \right] \right).$$

As the result, we received the following subclasses: $SC_1^4(Tr)$, $SC_6^4(Tr)$, $SC_7^4(Tr)$, $SC_8^4(Tr)$, $SC_9^4(Tr)$, $SC_2^5(Tr)$,and $SC_1^6(Tr)$. Let us join configurations $D_1$ and $D_2$, i.e.

$$D_3 = \left( Tr,\ ISD(Tr),\ N = [4,5,6],\ Q_a = \left[ include = \left[ Tr.p_1, Tr.f_1 \right],\ exclude = \left[ Tr.f_2, Tr.f_4 \right] \right], \right.$$
$$\left. Q_d = \left[ include = \left[\ \right],\ exclude = \left[\ \right] \right] \right).$$

As the result we received the following subclasses: $SC_1^4(Tr)$, $SC_7^4(Tr)$, $SC_8^4(Tr)$, $SC_2^5(Tr)$, and $SC_1^6(Tr)$. Now let us assume that we need to retrieve all semantically consistent subclasses of the class $Tr$, which have a cardinality from 4 to 6, and contain functional molecules $FM_1(Tr)$, $FM_3(Tr)$, for this purpose we need to set for Algorithm 3 the following configuration:

$$D_4 = \left( Tr,\ ISD(Tr),\ N = [4,5,6],\ Q_a = \left[ include = \left[\ \right],\ exclude = \left[\ \right] \right], \right.$$
$$\left. Q_d = \left[ include = \left[ FM_1(Tr), FM_3(Tr) \right],\ exclude = \left[\ \right] \right] \right).$$

In this case, the algorithm returned the following subclasses: $SC_7^4(Tr)$, $SC_8^4(Tr)$, $SC_9^4(Tr)$, $SC_2^5(Tr)$, $SC_4^5(Tr)$, $SC_5^5(Tr)$, $SC_6^5(Tr)$, $SC_1^6(Tr)$, $SC_2^6(Tr)$, and $SC_3^6(Tr)$. If we need to retrieve all semantically consistent subclasses of the class $Tr$, which have a cardinality from 4 to 6, and do not contain functional molecules $FM_2(Tr)$, $FM_4(Tr)$, for this purpose we need to set for Algorithm 3 the following configuration:

$$D_5 = \left( Tr,\ ISD(Tr),\ N = [4,5,6],\ Q_a = \left[ include = \left[\ \right],\ exclude = \left[\ \right] \right], \right.$$
$$\left. Q_d = \left[ include = \left[\ \right],\ exclude = \left[ FM_2(Tr), FM_4(Tr) \right] \right] \right).$$

In this case, the algorithm returned the following subclasses: $SC_1^4(Tr)$, $SC_6^4(Tr)$, $SC_7^4(Tr)$, $SC_8^4(Tr)$, $SC_9^4(Tr)$, $SC_2^5(Tr)$, and $SC_1^6(Tr)$. Let us join configurations $D_4$ and $D_5$, i.e.

$$D_6 = \Big( Tr,\ ISD(Tr),\ N = [4,5,6],\ Q_a = \big[ include = [\ ],\ exclude = [\ ] \big],$$
$$Q_d = \big[ include = \big[ FM_1(Tr), FM_3(Tr) \big],\ exclude = \big[ FM_2(Tr), FM_4(Tr) \big] \big] \Big).$$

As the result, we received the following subclasses: $SC_7^4(Tr)$, $SC_8^4(Tr)$, $SC_9^4(Tr)$, $SC_2^5(Tr)$, and $SC_1^6(Tr)$. Finally, let us join configurations $D_3$ and $D_6$, i.e.

$$D_7 = \Big( Tr,\ ISD(Tr),\ N = [4,5,6],\ Q_a = \big[ include = \big[ Tr.p_1, Tr.f_1 \big],\ exclude = \big[ Tr.f_2, Tr.f_4 \big] \big],$$
$$Q_d = \big[ include = \big[ FM_1(Tr), FM_3(Tr) \big],\ exclude = \big[ FM_2(Tr), FM_4(Tr) \big] \big] \Big).$$

In this case, the algorithm returned the following subclasses: $SC_7^4(Tr)$, $SC_8^4(Tr)$, $SC_2^5(Tr)$, and $SC_1^6(Tr)$. Let us consider the interpretation of the obtained results in more detail. As we can see, each of subclass $SC_7^4(Tr)$ and $SC_8^4(Tr)$ defines two points on a plane with the ability to get and set their coordinates, as well as compute the distance between them. According to the definition of the homogeneous class of objects $Tr$, they have a following representation:

$$\begin{aligned}
SC_7^4(Tr) = (&p_1 = (vertex_1, (v, Pt)),\\
&p_2 = (vertex_2, (v, Pt)),\\
&p_4 = is_a_triangle(vf_4(tr), v \in \{0,1\}),\\
&f_1 = get_vertex(tr, (i, \mathbb{Z}^+), Pt),\\
&f_3 = compute_side_length(tr, (vertex_a, Pt), (vertex_b, Pt), \mathbb{R}^+)),\\
SC_8^4(Tr) = (&p_1 = (vertex_1, (v, Pt)),\\
&p_3 = (vertex_3, (v, Pt)),\\
&p_4 = is_a_triangle(vf_4(tr), v \in \{0,1\}),\\
&f_1 = get_vertex(tr, (i, \mathbb{Z}^+), Pt),\\
&f_3 = compute_side_length(tr, (vertex_a, Pt), (vertex_b, Pt), \mathbb{R}^+)).
\end{aligned}$$

The subclass $SC_2^5(Tr)$ defines three points on a plane with the ability to get and set their coordinates, as well as compute the distance between any two of them. According to the definition of the class $Tr$, it has a following representation:

$$\begin{aligned}
SC_2^5(Tr) = (&p_1 = (vertex_1, (v, Pt)),\\
&p_2 = (vertex_2, (v, Pt)),\\
&p_3 = (vertex_3, (v, Pt)),\\
&f_1 = get_vertex(tr, (i, \mathbb{Z}^+), Pt),\\
&f_3 = compute_side_length(tr, (vertex_a, Pt), (vertex_b, Pt), \mathbb{R}^+)).
\end{aligned}$$

The subclass $SC_1^6(Tr)$ defines a triangle on a plane with the ability to get and set coordinates of its vertices, as well as compute the length of all its sides. According to the definition of the class $Tr$, it has a following representation:

$$
\begin{aligned}
SC_1^6(Tr) = (\ & p_1 = (vertex_1, (v, Pt)),\\
& p_2 = (vertex_2, (v, Pt)),\\
& p_3 = (vertex_3, (v, Pt)),\\
& p_4 = is\_a\_triangle(vf_4(tr), v \in \{0,1\}),\\
& f_1 = get\_vertex(tr, (i, \mathbb{Z}^+), Pt),\\
& f_3 = compute\_side\_length(tr, (vertex_a, Pt), (vertex_b, Pt), \mathbb{R}^+)).
\end{aligned}
$$

Therefore, we can conclude that decomposition of the homogeneous class of objects $Tr$ using Algorithm 3 generates semantically consistent subclasses, which represent the implicit or hidden knowledge within the domain of the class $Tr$. In addition, the algorithm performs the filtration of all constructed semantically consistent subclasses according to attribute query $Q_a$ and dependency query $Q_d$.

For some cases, Algorithm 3 can be improved by changing the filtration strategy, since a resolving of the decompositional SCP is also a kind of subclass filtration, depending on an attribute query, as well as a dependency query, the order of verification of their satisfiability can be changed. The main criterion for such modification of the algorithm is the estimation of search space reducing chain performed by a particular sequence of subclass filtration.

## 6. Conclusions

In this paper, we considered in detail the internal semantic dependencies of homogeneous classes of objects (structural and functional atoms and molecules) and how they affect the decomposition of the class. We defined the decomposition of the class as splitting the class into such subclasses, which do not contradict any internal semantic dependency. Since all possible subclasses of a homogeneous class of objects form a power set lattice, which is a complete lattice, using methods of formal concept analysis we constructed the corresponding concept lattices for all subclasses of the class, for all internal semantic dependencies of the class, and for all its semantically consistent subclasses. As the result, we found that in all three cases, constructed concept lattices contain a certain number of formal concepts with semantically inconsistent intents because the algorithms for the construction of concept lattices compute the part of extents via the intersection of extents which can be extracted from the formal context. At the same time, they do not consider the internal semantic dependencies of a class, which define corresponding restrictions to the creation of its semantically consistent subclasses. That restricts the usage of formal concept analysis for knowledge extraction and retrieval since it allows retrieval, inference, or usage of inconsistent concepts, which are unreal within a modeled domain.

To propose an alternative approach to knowledge extraction and retrieval via decomposition of homogeneous classes of objects, we improved the decomposition algorithm, which was proposed in [28], adding the additional filtering parameters, which help to reduce the search space and improve the performance. As the result, in the first stage, the algorithm extracts knowledge by constructing only semantically consistent subclasses of a homogeneous class of objects, which have a certain cardinality, via solving the corresponding constraint satisfaction problem defined based on the internal semantic dependencies of the class. In the second stage, the algorithm retrieves knowledge by filtration of constructed semantically consistent subclasses according to the attribute and dependency queries, which allow selecting only those subclasses, which include all desired attributes and dependencies and do not include undesired ones. We introduced the decomposition consistency coefficient, which allows us to estimate how much the algorithm can reduce the search space for knowledge extraction and retrieval, avoiding the consideration of all semantically inconsistent subclasses of the class. To demonstrate the possible application of the algorithm, we considered seven different scenarios of how the homogeneous class of objects, which define a triangle on a plane, can be decomposed for knowledge extraction and retrieval. In all cases, the algorithm extracted and retrieved subclasses of the class, which are semantically consistent within a modeled domain and satisfy all restrictions and filters.

In addition, we developed two algorithms for the decomposition of objects, which adapt generated objects for their usage in the run-time or storing them in a database. Both algorithms are based on the decomposition of a homogeneous class of objects and guarantee that all objects of any semantically consistent subclass of the class, generated during the decomposition, will be also semantically consistent ones as their class. The proposed approach provides an opportunity to perform knowledge extraction via the decomposition of the homogeneous classes of objects, as well as the decomposition of their objects. To demonstrate the possible application of these algorithms, we provided corresponding examples of decomposition for objects of subclasses of different cardinality. However, despite all mentioned advantages of the developed algorithms, they require future analysis, improvement, and optimization.

## 7. Acknowledgements

This research has been supported by the National Academy of Science of Ukraine (project 0121U111944 Development of Methods and Tools for Construction of Domain-Oriented Intelligent Software Systems Based on Object-Oriented Dynamic Networks).

## References

[1] S. W. Ambler, Agile Database Techniques: Effective Strategies for the Agile Software Developer, John Willey & Sons, Ltd., Indianapolis, IN, USA, 2003.

[2] S. W. Ambler, The Object Primer: Agile Model-Driven Development with UML 2.0, 3rd ed., Cambridge University Press, New York, NY, USA, 2004.

[3] K. R. Apt, Principles of Constraint Programming, Cambridge University Press, New York, NY, USA, 2003. doi: 10.1017/CBO9780511615320.

[4] J. Baixeries, M. Kaytoue, A. Napoli, Characterizing functional dependencies in formal concept analysis with pattern structures, Annals of Mathematics and Artificial Intelligence. 72 (2014) 129-149. doi: 10.1007/s10472-014-9400-3.

[5] J. Baixeries, M. Kaytoue, A. Napoli, Characterization of Database Dependencies with FCA and Pattern Structures, in: D. I. Ignatov et al. (Eds.), Analysis of Images, Social Networks and Texts. AIST 2014, volume 436 of *CCIS*, Springer, Cham., 2014, pp. 3-14. doi: 10.1007/978-3-319-12580-0_1.

[6] J. Baixeries, V. Codocedo, M. Kaytoue, A. Napoli, Computing Dependencies Using FCA, in: R. Missaoui, L. Kwuida, and T. Abdessalem (Eds.), Complex Data Analytics with Formal Concept Analysis, Springer Cham., Switzerland AZ, 2022. doi: 10.1007/978-3-030-93278-7_6.

[7] C. Carpineto, G. Romano. Concept data analysis: Theory and application, John Willey & Sons, Ltd., New York, NY, USA, 2004.

[8] L. Caruccio, V. Deufemia, G. Polese. Relaxed Functional Dependencies – A Survey of Approaches, IEEE Transactions on Knowledge and Data Engineering, 28 (2016) 147-165. doi: 10.1109/TKDE.2015.2472010.

[9] V. Codocedo, I. Lykourentzou, H. Astudillo, A. Napoli, Using pattern structures to support information retrieval with Formal Concept Analysis, in: Proceedings of the International Workshop "What can FCA do for Artificial Intelligence?" (FCA4AI at IJCAI 2013), volume 1058, Beijing, China, 2013, pp. 15-24.

[10] V. Codocedo, I. Lykourentzou, and A. Napoli, A semantic approach to concept lattice-based information retrieval, Annals of Mathematics and Artificial Intelligence, 72 (2014) 169-195. doi: 10.1007/s10472-014-9403-0.

[11] V. Codocedo, A. Napoli, Formal Concept Analysis and Information Retrieval – A Survey, in: J. Baixeries, C. Sacarea, and M. Ojeda-Aciego (Eds.), Formal Concept Analysis. ICFCA 2015, volume 9113 of *LNCS*, Springer Cham., 2015, pp. 61-77. doi: 10.1007/978-3-319-19545-2_4.

[12] R. Dechter, Constraint Processing. Morgan Kaufmann Publishers, San Francisco, CA, USA 2003. doi: 10.1016/B978-1-55860-890-0.X5000-2.

[13] U. Dekel, Applications of Concept Lattices to Code Inspection and Review, in: Proceedings of the Israeli Workshop on Programming Languages and Development Environments. IBM Haifa Research Lab, 2002.
[14] U. Dekel, Y. Gil, Revealing Class Structure with Concept Lattices, in: Proceedings of the 10th Working Conference on Reverse Engineering (WCRE'03). Victoria, BC, Canada, 2003, pp. 353-363. doi: 10.1109/WCRE.2003.1287267.
[15] S. Dominich, The Modern Algebra of Information Retrieval, Springer, Berlin, Heidelberg, 2008. doi: 10.1007/987-3-540-77659-8.
[16] D. Dubois, H. Prade, J.-P. Rossazza, Vagueness, typicality, and uncertainty in class hierarchies, International Journal of Intelligent Systems, 6 (1991) 167-183. doi: 10.1002/int.4550060205.
[17] F. Fkih, M. N. Omri, IRAFCA: an O(n) information retrieval algorithm based on formal concept analysis, Knowledge and Information Systems, 48 (2016) 465-491. doi: 10.1007/s10115-015-0876-x.
[18] F. Fkih, M. N. Omri, FCA_Retrieval: A Multi-operator Algorithm for Information Retrieval from Binary Concept Lattice, in: Proceedings of the 32nd Pacific Asia Conference on Language, Information and Computation, PACLIC 2018, Hong Kong, 2018, pp. 164-171.
[19] E. C. Freuder, A. K. Mackworth, Constraint satisfaction: an emerging paradigm, in: F. Rossi, P. van Beek, and T. Walsh (Eds.), Handbook of Constraint Programming, Elsevier, Amsterdam, 2006.
[20] B. Ganter, R. Wille, Formal Concept Analysis: Mathematical Foundations, Springer, Berlin, Heidelberg, 1999. doi: 10.1007/978-3-642-59830-2.
[21] B. Ganter, S. Rudolph, G. Stumme, Explaining Data with Formal Concept Analysis, in: M. Krotzsch, and D. Stepanova (Eds.), Reasoning Web. Explainable Artificial Intelligence, volume 11810 of *LNCS*, Springer Cham., 2019, pp. 153-195. doi: 10.1007/978-3-030-31423-1_5.
[22] R. Goldblatt, Topoi. The Categorial Analysis of Logic. Revised edition, volume 98 of Studies in Logic and the Foundations of Mathematics, Elsevier Science Publishers B.V., Amsterdam, 1984.
[23] P. Joshi, R. J. Joshi, Concept Analysis for Class Cohesion, in: 13th European Conference on Software Maintenance and Reengineering, Kaiserslautern, Germany, 2009, pp. 237-240. doi: 10.1109/CSMR.2009.54.
[24] L. Kwuida, R. Missaoui, A. Balamane, J. Vaillancourt, Generalized pattern extraction from concept lattices, Annals of Mathematics and Artificial Intelligence. 72 (2014) 151-168. doi: 10.1007/s10472-014-9411-0.
[25] N. Messai, M.-D. Devignes, A. Napoli, M. Smail-Tabbone, Extending Attribute Dependencies for Lattice-Based Querying and Navigation, in P. Eklund, and O. Haemmerle (Eds.), Conceptual Structures: Knowledge Visualization and Reasoning. ICCS 2008, volume 5113 of *LNCS*, Springer, Berlin, Heidelberg, 2008, pp. 189-202. doi: 10.1007/978-3-540-70569-3_13.
[26] D. O. Terletskyi, O. I. Provotar, Object-oriented dynamic networks, in: G. Setlak, and, K. Markov (Eds.), Computational Models for Business and Engineering Domains, volume 30 of International Book Series Information Science and Computing, ITHEA, Rzeszow-Sofia, 2014.
[27] D. O. Terletskyi, O. I. Provotar, Mathematical foundations for designing and development of intelligent systems of information analysis, in: Proceedings of 9th International and Practical Conference on Programming, UrkPROG'2014, Kyiv, Ukraine, 2014, pp. 233-241.
[28] D. O. Terletskyi, Run-Time Class Generation: Algorithm for Decomposition of Homogeneous Classes. in: A. Lopata, R. Butkiene, D. Gudoniene, and V. Sukacke (Eds.), Information and Software Technologies. ICIST 2020, volume 1238 of *CCIS*, Springer Cham., Switzerland AZ, 2020, pp. 234-254. doi: 10.1007/978-3-030-59506-7_20.
[29] M. Tiwari, P. Joshi, Method Cohesion Analysis through Concept Lattices, ACM SIGSOFT Software Engineering Notes, 37 (2012) 1-4. doi: 10.1145/180921.2180924.
[30] P. Tonella, Using a Concept Lattice of Decomposition Slices for Program Understanding and Impact Analysis, IEEE Transactions on Software Engineering. 29 (2003) 495-509. doi: 10.1109/TSE.2003.1205178.

[31] R. Wille, Methods of Conceptual Knowledge Processing, in: R. Missaoui, J. Schmid (Eds.), Formal Concept Analysis. ICFCA 2006, volume 3874 of *LNCS*, Springer, Berlin, Heidelberg, 2006, pp. 1-29. doi: 10.1007/11671404_1.
[32] H. Yao, H. J. Hamilton, Mining functional dependencies from data, Data Mining and Knowledge Discovery, 16 (2008) 197-219. doi: 10.1007/s10618-007-0083-9.